Lecture 29

Monday, Dec. 10, 2001

Primitive Operations of Transactions

- INPUT(X)
  - read element X to memory buffer
- READ(X,t)
  - copy element X to transaction local variable t
- WRITE(X,t)
  - copy transaction local variable t to element X
- OUTPUT(X)
  - write element X to disk

Example

READ(A,t); t := t*2;WRITE(A,t)
READ(B,t); t := t*2;WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
REAT(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

The Log

- An append-only file containing log records
- Note: multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo some transaction that didn't commit
  - Undo other transactions that didn't commit

Undo Logging

Log records

- <START T>
  - transaction T has begun
- <COMMIT T>
  - T has committed
- <ABORT T>
  - T has aborted
- <T,X,v>
  - T has updated element X, and its *old* value was v

Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before X is written to disk

U2: If T commits, then <COMMIT T> must be written to disk only after all changes by T are written to disk

- Hence: OUTPUTs are done *early*

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						<COMMIT T>

Recovery with Undo Log

After system's crash, run recovery manager

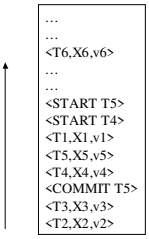
- Idea 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>…. = yes
  - <START T>….<ABORT T>……. = yes
  - <START T>…… = no
- Idea 2. Undo all modifications by incompleted transactions

Recovery with Undo Log

Recovery manager:

- Read log from the end; cases:
  - <COMMIT T>: mark T as completed
  - <ABORT T>: mark T as completed
  - <T,X,v>: if T is not completed
    then write X=v to disk
    else ignore
  - <START T>: ignore

Recovery with Undo Log



Recovery with Undo Log

- Note: all undo commands are *idempotent*
  - If we perform them a second time, no harm is done
  - E.g. if there is a system crash during recovery, simply restart recovery from scratch

Recovery with Undo Log

When do we stop reading the log ?

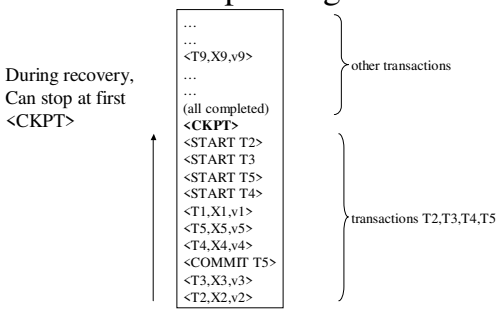
- We cannot stop until we reach the beginning of the log file
- This is impractical
- Better idea: use checkpointing

Checkpointing

Checkpoint the database periodically

- Stop accepting new transactions
- Wait until all curent transactions complete
- Flush log to disk
- Write a <CKPT> log record, flush
- Resume transactions

Undo Recovery with Checkpointing



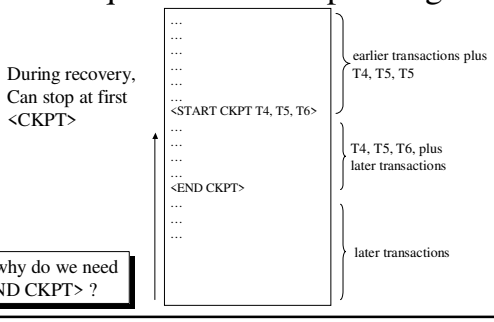
Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- =nonquiescent checkpointing

Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Continue normal operation
- When all of T1,…,Tk have completed, write <END CKPT>

Undo Recovery with Nonquiescent Checkpointing



Q: why do we need <END CKPT> ?

Redo Logging

Log records

- <START T> = transaction T has begun
- <COMMIT T> = T has committed
- <ABORT T>= T has aborted
- <T,X,v>= T has updated element X, and its *new* value is v

Redo-Logging Rules

R1: If T modifies X, then both <T,X,v> and <COMMIT T> must be written to disk before X is written to disk

- Hence: OUTPUTs are done *late*

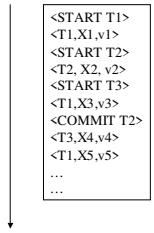
Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Recovery with Redo Log

After system's crash, run recovery manager

- Step 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>…. = yes
  - <START T>….<ABORT T>……. = yes
  - <START T>…… = no
- Step 2. Read log from the beginning, redo all updates of *committed* transactions

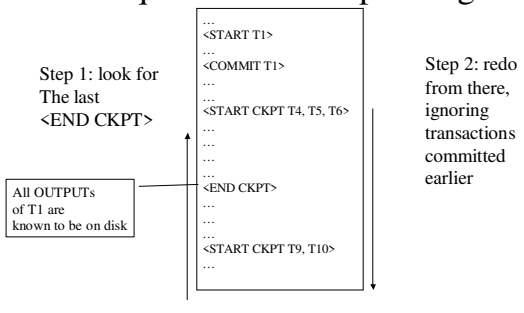
Recovery with Redo Log



Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Flush to disk all blocks of committed transactions (*dirty blocks*), while continuing normal operation
- When all blocks have been written, write <END CKPT>

Redo Recovery with Nonquiescent Checkpointing



Comparison Undo/Redo

- Undo logging:
  - OUTPUT must be done early
  - If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don't need to undo)
- Redo logging
  - OUTPUT must be done late
  - If <COMMIT T> is not seen, T definitely has not written any of its data to disk (hence there is not dirty data on disk)
- Would like more flexibility on when to OUTPUT: undo/redo logging (next)

Undo/Redo Logging

Log records, only one change

- <T,X,u,v>= T has updated element X, its *old* value was u, and its *new* value is v

Undo/Redo-Logging Rule

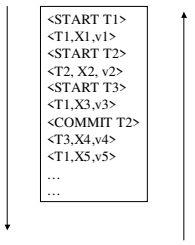
UR1: If T modifies X, then <T,X,u,v> must be written to disk before X is written to disk

Note: we are free to OUTPUT early or late (I.e. before or after <COMMIT T>)

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
OUTPUT(A)	16	16	16	16	8	
						<COMMIT T>
OUTPUT(B)	16	16	16	16	16	

Recovery with Undo/Redo Log

After system's crash, run recovery manager

- Redo all committed transaction, top-down
- Undo all uncommitted transactions, bottom-up

Recovery with Redo Log