

CSE 444: Database Internals

Lectures 18-19
Transactions: Aries

Readings

- Material in today's lecture NOT in the book
- Instead, read [Sections 1, 2.2, and 3.2](#) of:
Michael J. Franklin. **Concurrency Control and Recovery**. The Handbook of Computer Science and Engineering, A. Tucker, ed., CRC Press, Boca Raton, 1997.
 - Posted on course website (below the calendar)
 - Other sections are also worth reading

Buffer Manager Policies

- **STEAL or NO-STEAL**
 - Can an update made by an uncommitted transaction overwrite the most recent committed value of a data item on disk?
- **FORCE or NO-FORCE**
 - Should all updates of a transaction be forced to disk before the transaction commits?
- Easiest for recovery: NO-STEAL/FORCE
- Highest performance: STEAL/NO-FORCE

Comparison Undo/Redo

- **Undo logging:**
 - OUTPUT must be done early Steal/Force
 - If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don't need to redo) – inefficient
- **Redo logging**
 - OUTPUT must be done late No-Steal/No-Force
 - 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, no need to undo) – inflexible
- Would like more flexibility on when to OUTPUT:
undo/redo logging (next) Steal/No-Force

Aries Recovery Algorithm

- An UNDO/REDO log with lots of clever details

Write-Ahead Log

- Enables the use of STEAL and NO-FORCE
- **Log: append-only file containing log records**
- For every update, commit, or abort operation
 - Write physiological log record (more later)
 - Note: multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
 - Redo some transaction that did commit
 - Undo other transactions that didn't commit

Write-Ahead Log

- All log records pertaining to a **page** are written to disk **before** the page is **overwritten** on disk
- All log records for **transaction** are written to disk **before** the transaction is considered **committed**
 - Why is this faster than FORCE policy?
- **Committed transaction**: transactions whose commit log record has been written to disk

Magda Balazinska - CSE 444, Spring 2012

7

Log Granularity

Two basic types of log records for update operations

- **Physical log records**
 - Position on a particular page where update occurred
 - Both before and after image for undo/redo logs
 - Benefits: Idempotent & updates are fast to redo/undo
- **Logical log records**
 - Record only high-level information about the operation
 - Benefit: Smaller log
 - BUT difficult to implement because crashes can occur in the middle of an operation

Magda Balazinska - CSE 444, Spring 2012

8

Granularity in ARIES

- **Physiological logging**
 - Log records refer to a single page
 - But record logical operation within the page
- **Page-oriented logging for REDO**
 - Necessary since can crash in middle of complex op.
- **Logical logging for UNDO**
 - Enables **tuple-level locking!**
 - Must do logical undo because ARIES will only undo loser transactions (this also facilitates ROLLBACKS)

Magda Balazinska - CSE 444, Spring 2012

9

ARIES Method

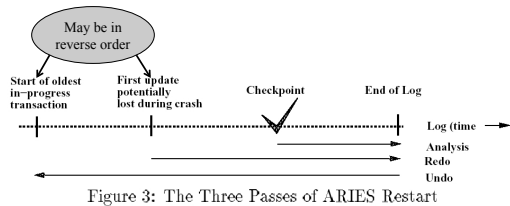
Recovery from a system crash is done in 3 passes:

1. **Analysis pass**
 - Figure out what was going on at time of crash
 - List of dirty pages and active transactions
2. **Redo pass (repeating history principle)**
 - Redo all operations, even for transactions that will not commit
 - Get back to state at the moment of the crash
3. **Undo pass**
 - Remove effects of all uncommitted transactions
 - Log changes during undo in case of another crash during undo

Magda Balazinska - CSE 444, Spring 2012

10

ARIES Method Illustration



[Franklin97]

Magda Balazinska - CSE 444, Spring 2012

11

ARIES Data Structures

- **Active Transactions Table**
 - Lists all running transactions (active transactions)
 - For each txn: **lastLSN** = most recent update by transaction
- **Dirty Page Table**
 - Lists all dirty pages
 - For each dirty page: **recoveryLSN (recLSN)** = first LSN that caused page to become dirty
- **Write Ahead Log** contains log records
 - LSN, **prevLSN** = previous LSN for same transaction
 - other attributes

Magda Balazinska - CSE 444, Spring 2012

12

ARIES Data Structures

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Log

LSN	prevLSN	transID	pageID	Log entry
101	-	T100	P7	
102	-	T200	P5	
103	102	T200	P6	
104	101	T100	P5	

Active transactions

transID	lastLSN
T100	104
T200	103

Buffer Pool

P5 PageLSN=104	P6 PageLSN=103	P7 PageLSN=101

The LSN

- Each log entry receives a unique *Log Sequence Number*, LSN
 - The LSN is written in the log entry
 - Entries belonging to the same transaction are chained in the log via **prevLSN**
 - LSN's help us find the end of a circular log file:

After crash, log file = (22, 23, 24, 25, 26, 18, 19, 20, 21)
Where is the end of the log ? 18

Magda Balazinska - CSE 444, Spring 2012

14

ARIES Method Details

- Steps under normal operations
 - Add log record
 - Update transactions table
 - Update dirty page table
 - Update pageLSN

Magda Balazinska - CSE 444, Spring 2012

15

ARIES Method

- More details and long example on the board
- Please TAKE NOTES!

Magda Balazinska - CSE 444, Spring 2012

16

Checkpoints

- Write into the log
 - Entire **active transactions table**
 - Entire **dirty page table**
- Very fast ! No waiting, no END CKPT
- But, effectiveness is limited by dirty pages
 - Solution: A background process periodically sends dirty pages to disk

Magda Balazinska - CSE 444, Spring 2012

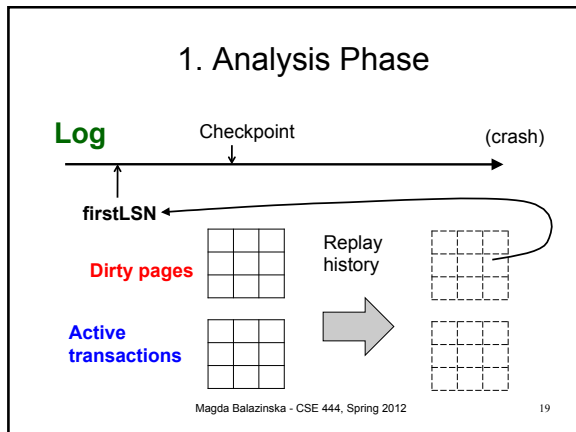
17

1. Analysis Phase

- Goal
 - Determine point in log where to start REDO
 - Determine set of dirty pages when crashed
 - Conservative estimate of dirty pages
 - Identify active transactions when crashed
- Approach
 - Rebuild **active transactions table** and **dirty pages table**
 - Reprocess the log from the beginning (or checkpoint)
 - Only update the two data structures
 - Compute: **firstLSN** = smallest of all **recoveryLSN**

Magda Balazinska - CSE 444, Spring 2012

18



2. Redo Phase

Main principle: replay history

- Process Log forward, starting from **firstLSN**
- Read every log record, sequentially
- Redo actions are not recorded in the log
- Needs the **Dirty Page Table**

Magda Balazinska - CSE 444, Spring 2012 20

2. Redo Phase: Details

For each **Log** entry record LSN

- If affected page is not in **Dirty Page Table** then **do not update**
- If **recoveryLSN** > LSN, then **no update**
- Read page from disk;
If **pageLSN** > LSN, then **no update**
- Otherwise perform update

Magda Balazinska - CSE 444, Spring 2012 21

3. Undo Phase

Main principle: "logical" undo

- Start from the end of the log, move backwards
- Read only affected log entries
- Undo actions *are* written in the Log as special entries: CLR (Compensating Log Records)
- CLR's are redone, but never undone

Magda Balazinska - CSE 444, Spring 2012 22

3. Undo Phase: Details

- "Loser transactions" = uncommitted transactions in **Active Transactions Table**
- **ToUndo** = set of **lastLSN** of loser transactions
- While **ToUndo** not empty:
 - Choose most recent (largest) LSN in **ToUndo**
 - If LSN = regular record: undo; write a CLR where CLR.undoNextLSN = LSN.prevLSN; if LSN.prevLSN not null, insert in **ToUndo** otherwise, write <END TRANSACTION> in log
 - If LSN = CLR record: (don't undo !)
if CLR.undoNextLSN not null, insert in **ToUndo** otherwise, write <END TRANSACTION> in log

Magda Balazinska - CSE 444, Spring 2012 23

