



# ARIES Recovery Algorithm

*ARIES: A Transaction Recovery Method  
Supporting Fine Granularity Locking and Partial  
Rollback Using Write-Ahead Logging*

C. Mohan, D. Haderle, B. Lindsay,  
H. Pirahesh, and P. Schwarz

ACM Transactions on Database Systems, 17(1), 1992

Slides prepared by  
S. Sudarshan



©Silberschatz, Korth and Sudarshan

1



# Recovery Scheme Metrics

- Concurrency
- Functionality
- Complexity
- Overheads:
  - ★ Space and I/O (Seq and random) during Normal processing and recovery
- Failure Modes:
  - ★ transaction/process, system and media/device



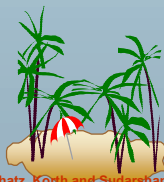
©Silberschatz, Korth and Sudarshan

2



# Key Features of Aries

- Physical Logging, and
- Operation logging
  - ★ e.g. Add 5 to A, or insert K in B-tree B
- Page oriented redo
  - ★ recovery independence amongst objects
- Logical undo (may span multiple pages)
- WAL + Inplace Updates



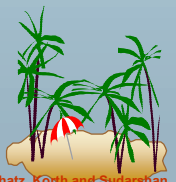
©Silberschatz, Korth and Sudarshan

3



# Key Aries Features (contd)

- Transaction Rollback
  - ★ Total vs partial (up to a savepoint)
  - ★ Nested rollback - partial rollback followed by another (partial/total) rollback
- Fine-grain concurrency control
  - ★ supports tuple level locks on records, and key value locks on indices



©Silberschatz, Korth and Sudarshan

4



## More Aries Features

- Flexible storage management
  - ★ Physiological redo logging:
    - logical operation within a single page
    - no need to log intra-page data movement for compaction
    - LSN used to avoid repeated redos (more on LSNs later)
- Recovery independence
  - ★ can recover some pages separately from others
- Fast recovery and parallelism



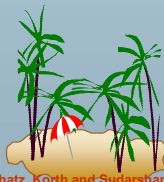
## Latches and Locks

- Latches
  - ★ used to guarantee physical consistency
  - ★ short duration
  - ★ no deadlock detection
  - ★ direct addressing (unlike hash table for locks)
    - often using atomic instructions
    - latch acquisition/release is much faster than lock acquisition/release
- Lock requests
  - ★ conditional, instant duration, manual duration, commit duration



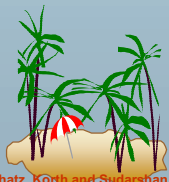
## Buffer Manager

- Fix, unfix and fix\_new (allocate and fix new pg)
- Aries uses **steal policy** - uncommitted writes may be output to disk (contrast with **no-steal** policy)
- Aries uses **no-force** policy (updated pages need not be forced to disk before commit)
- dirty page: buffer version has updated not yet reflected on disk
  - ★ dirty pages written out in a continuous manner to disk



## Buffer Manager (Contd)

- BCB: buffer control blocks
  - ★ stores page ID, dirty status, latch, fix-count
- Latching of pages = latch on buffer slot
  - ★ limits number of latches required
  - ★ but page must be fixed before latching





## Some Notation

- LSN: Log Sequence Number
  - ★ = logical address of record in the log
- Page LSN: stored in page
  - ★ LSN of most recent update to page
- PrevLSN: stored in log record
  - ★ identifies previous log record for that transaction
- Forward processing (normal operation)
- Normal undo vs. restart undo



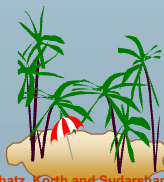
## Compensation Log Records

- CLR: redo only log records
- Used to record actions performed during transaction rollback
  - ★ one CLR for each normal log record which is undone
- CLR: have a field **UndoNxtLSN** indicating which log record is to be undone next
  - avoids repeated undos by bypassing already undo records
    - needed in case of restarts during transaction rollback)
  - in contrast, IBM IMS may repeat undos, and AS400 may even undo undos, then redo the undos



## Normal Processing

- Transactions add log records
- Checkpoints are performed periodically
  - ★ contains
    - Active transaction list,
    - LSN of most recent log records of transaction, and
    - List of dirty pages in the buffer (and their recLSNs)
      - to determine where redo should start



## Recovery Phases

- Analysis pass
  - ★ forward from last checkpoint
- Redo pass
  - ★ forward from RedoLSN, which is determined in analysis pass
- Undo pass
  - ★ backwards from end of log, undoing incomplete transactions





## Analysis Pass

- RedoLSN = min(LSNs of dirty pages recorded in checkpoint)
  - ★ if no dirty pages, RedoLSN = LSN of checkpoint
  - ★ pages dirtied later will have higher LSNs)
- scan log forwards from last checkpoint
  - ★ find transactions to be rolled back ("loser" transactions)
  - ★ find LSN of last record written by each such transaction



13

©Silberschatz, Korth and Sudarshan



## Redo Pass

- Repeat history, scanning forward from RedoLSN
  - ★ for all transactions, even those to be undone
  - ★ perform redo only if page\_LSN < log records LSN
  - ★ no locking done in this pass



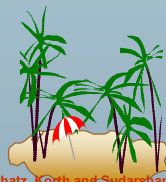
14

©Silberschatz, Korth and Sudarshan



## Undo Pass

- Single scan backwards in log, undoing actions of "loser" transactions
  - ★ for each transaction, when a log record is found, use prev\_LSN fields to find next record to be undone
  - ★ can skip parts of the log with no records from loser transactions
  - ★ don't perform any undo for CLR (note: UndoNxtLSN for CLR indicates next record to be undone, can skip intermediate records of that transactions)

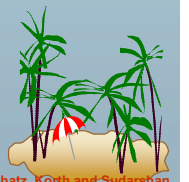


15

©Silberschatz, Korth and Sudarshan



## Data Structures Used in Aries



16

©Silberschatz, Korth and Sudarshan



## Log Record Structure

### ■ Log records contain following fields

- ★ LSN
- ★ Type (CLR, update, special)
- ★ TransID
- ★ PrevLSN (LSN of prev record of this txn)
- ★ PageID (for update/CLRs)
- ★ UndoNxtLSN (for CLRs)
  - indicates which log record is being compensated
  - on later undos, log records upto UndoNxtLSN can be skipped
- ★ Data (redo/undo data); can be physical or logical



17

©Silberschatz, Korth and Sudarshan



## Transaction Table

### ■ Stores for each transaction:

- ★ TransID, State
- ★ LastLSN (LSN of last record written by txn)
- ★ UndoNxtLSN (next record to be processed in rollback)

### ■ During recovery:

- ★ initialized during analysis pass from most recent checkpoint
- ★ modified during analysis as log records are encountered, and during undo



18

©Silberschatz, Korth and Sudarshan



## Dirty Pages Table

### ■ During normal processing:

- ★ When page is fixed with intention to update
  - Let L = current end-of-log LSN (the LSN of next log record to be generated)
  - if page is not dirty, store L as RecLSN of the page in dirty pages table
- ★ When page is flushed to disk, delete from dirty page table
- ★ dirty page table written out during checkpoint
- ★ (Thus RecLSN is LSN of earliest log record whose effect is not reflected in page on disk)



19

©Silberschatz, Korth and Sudarshan



## Dirty Page Table (contd)

### ■ During recovery

- ★ load dirty page table from checkpoint
- ★ updated during analysis pass as update log records are encountered



20

©Silberschatz, Korth and Sudarshan



## Normal Processing Details



21

©Silberschatz, Korth and Sudarshan



## Updates

- Page latch held in X mode until log record is logged
  - ★ so updates on same page are logged in correct order
  - ★ page latch held in S mode during reads since records may get moved around by update
  - ★ latch required even with page locking if dirty reads are allowed
- Log latch acquired when inserting in log



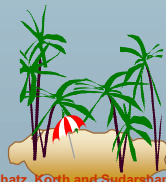
22

©Silberschatz, Korth and Sudarshan



## Updates (Contd.)

- Protocol to avoid deadlock involving latches
  - ★ deadlocks involving latches and locks were a major problem in System R and SQL/DS
  - ★ transaction may hold at most two latches at-a-time
  - ★ must never wait for lock while holding latch
    - if both are needed (e.g. Record found after latching page):
    - release latch before requesting lock and then reacquire latch (and recheck conditions in case page has changed inbetween).  
Optimization: conditional lock request
  - ★ page latch released before updating indices
    - data update and index update may be out of order



23

©Silberschatz, Korth and Sudarshan



## Split Log Records

- Can split a log record into undo and redo parts
  - ★ undo part must go first
  - ★ page\_LSN is set to LSN of redo part



24

©Silberschatz, Korth and Sudarshan



## Savepoints

- Simply notes LSN of last record written by transaction (up to that point) - denoted by SaveLSN
- can have multiple savepoints, and rollback to any of them
- deadlocks can be resolved by rollback to appropriate savepoint, releasing locks acquired after that savepoint



25

©Silberschatz, Korth and Sudarshan



## Rollback

- Scan backwards from last log record of txn
  - (last log record of txn = transTable[TransID].UndoNxtLSN
  - ★ if log record is an update log record
    - undo it and add a CLR to the log
  - ★ if log record is a CLR
    - then UndoNxt = LogRec.UnxoNxtLSN
    - else UndoNxt = LogRec.PrevLSN
  - ★ next record to process is UndoNxt; stop at SaveLSN or beginning of transaction as required



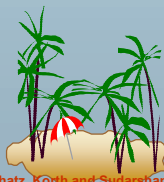
26

©Silberschatz, Korth and Sudarshan



## More on Rollback

- Extra logging during rollback is bounded
  - ★ make sure enough log space is available for rollback in case of system crash, else BIG problem
- In case of 2PC, if in-doubt txn needs to be aborted, rollback record is written to log then rollback is carried out



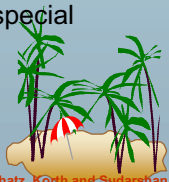
27

©Silberschatz, Korth and Sudarshan



## Transaction Termination

- prepare record is written for 2PC
  - ★ locks are noted in prepare record
- prepare record also used to handle non-undoable actions e.g. deleting file
  - these **pending actions** are noted in prepare record and executed only after actual commit
- end record written at commit time
  - ★ pending actions are then executed and logged using special redo-only log records
- end record also written after rollback



28

©Silberschatz, Korth and Sudarshan



## Checkpoints

- begin\_chkpt record is written first
- transaction table, dirty\_pages table and some other file mgmt information are written out
- end\_chkpt record is then written out
  - ★ for simplicity all above are treated as part of end\_chkpt record
- LSN of begin\_chkpt is then written to **master** record in well known place on stable storage
- incomplete checkpoint
  - ★ if system crash before end\_chkpt record is written



29

©Silberschatz, Korth and Sudarshan



## Checkpoint (contd)

- Pages need not be flushed during checkpoint
  - ★ are flushed on a continuous basis
- Transactions may write log records during checkpoint
- Can copy dirty\_page table fuzzily (hold latch, copy some entries out, release latch, repeat)



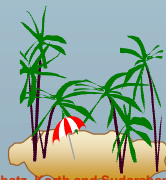
30

©Silberschatz, Korth and Sudarshan



## Restart Processing

- Finds checkpoint begin using master record
- Do restart\_analysis
- Do restart\_redo
  - ★ ... some details of dirty page table here
- Do restart\_undo
- reacquire locks for prepared transactions
- checkpoint



31

©Silberschatz, Korth and Sudarshan



## Result of Analysis Pass

- Output of analysis
  - ★ transaction table
    - including UndoNxtLSN for each transaction in table
  - ★ dirty page table: pages that were potentially dirty at time of crash/shutdown
  - ★ RedoLSN - where to start redo pass from
- Entries added to dirty page table as log records are encountered in forward scan
  - ★ also some special action to deal with OS file deletes
- This pass can be combined with redo pass!



32

©Silberschatz, Korth and Sudarshan





## Redo Pass

- Scan forward from RedoLSN
  - ★ If log record is an update log record, AND is in dirty\_page\_table AND LogRec.LSN  $\geq$  RecLSN of the page in dirty\_page\_table
  - ★ then if pageLSN  $<$  LogRec.LSN then perform redo; else just update RecLSN in dirty\_page\_table
- Repeats history: redo even for loser transactions (some optimization possible)



33

©Silberschatz, Korth and Sudarshan



## More on Redo Pass

- Dirty page table details
  - ★ dirty page table from end of analysis pass (restart dirty page table) is used and set in redo pass (and later in undo pass)
- Optimizations of redo
  - ★ Dirty page table info can be used to pre-read pages during redo
  - ★ Out of order redo is also possible to reduce disk seeks



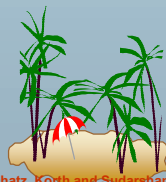
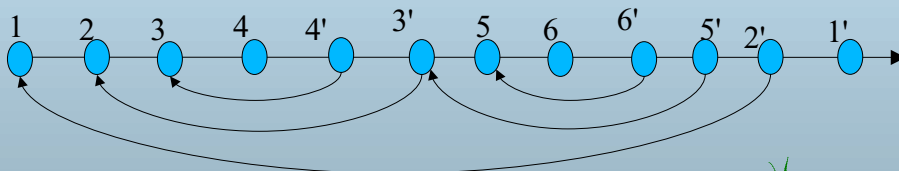
34

©Silberschatz, Korth and Sudarshan



## Undo Pass

- Rolls back loser transaction in reverse order in single scan of log
  - ★ stops when all losers have been fully undone
  - ★ processing of log records is exactly as in single transaction rollback



35

©Silberschatz, Korth and Sudarshan



## Undo Optimizations

- Parallel undo
  - ★ each txn undone separately, in parallel with others
  - ★ can even generate CLRs and apply them separately, in parallel for a single transaction
- New txns can run even as undo is going on:
  - ★ reacquire locks of loser txns before new txns begin
  - ★ can release locks as matching actions are undone



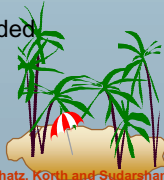
36

©Silberschatz, Korth and Sudarshan



## Undo Optimization (Contd)

- If pages are not available (e.g media failure)
  - ★ continue with redo recovery of other pages
    - once pages are available again (from archival dump) redos of the relevant pages must be done first, before any undo
  - ★ for physical undos in undo pass
    - we can generate CLR's and apply later; new txns can run on other pages
  - ★ for logical undos in undo pass
    - postpone undos of loser txns if the undo needs to access these pages - "stopped transaction"
    - undo of other txns can proceed; new txns can start provided appropriate locks are first acquired for loser txns



37

©Silberschatz, Korth and Sudarshan



## Transaction Recovery

- Loser transactions can be restarted in some cases
  - e.g. Mini batch transactions which are part of a larger transaction



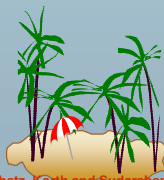
38

©Silberschatz, Korth and Sudarshan



## Checkpoints During Restart

- Checkpoint during analysis/redo/undo pass
  - ★ reduces work in case of crash/restart during recovery
    - (why is Mohan so worried about this!)
  - ★ can also flush pages during redo pass
    - RecLSN in dirty page table set to current last-processed-record



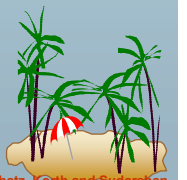
39

©Silberschatz, Korth and Sudarshan



## Media Recovery

- For archival dump
  - ★ can dump pages directly from disk (bypass buffer, no latching needed) or via buffer, as desired
    - this is a fuzzy dump, not transaction consistent
  - ★ begin\_chkpt location of most recent checkpoint completed before archival dump starts is noted
    - called image copy checkpoint
    - redoLSN computed for this checkpoint and noted as media recovery redo point



40

©Silberschatz, Korth and Sudarshan



## Media Recovery (Contd)

- To recover parts of DB from media failure
  - ★ failed parts if DB are fetched from archival dump
  - ★ only log records for failed part of DB are reapplied in a redo pass
  - ★ inprogress transactions that accessed the failed parts of the DB are rolled back
- Same idea can be used to recover from page corruption
  - ★ e.g. Application program with direct access to buffer crashes before writing undo log record



## Nested Top Actions

- Same idea as used in logical undo in our advanced recovery mechanism
  - ★ used also for other operations like creating a file (which can then be used by other txns, before the creator commits)
  - ★ updates of nested top action commit early and should not be undone
- Use dummy CLR to indicate actions should be skipped during undo

