UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems

Lecture 17: ARIES

Xiangyao Yu

11/2/2022

Today's Paper: ARIES

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

C. MOHAN
IBM Almaden Research Center
and
DON HADERLE
IBM Santa Teresa Laboratory
and
BRUCE LINDSAY, HAMID PIRAHESH and PETER SCHWARZ
IBM Almaden Research Center

In this paper we present a simple and efficient method, called ARIES (*Algorithm for Recovery and Isolation Exploiting Semantics*), which supports partial rollbacks of transactions, fine-granularity (e.g., record) locking and recovery using write-ahead logging (WAL). We introduce the paradigm of *repeating history* to redo all missing updates *before* performing the rollbacks of the loser transactions during restart after a system failure. ARIES uses a log sequence number in each page to correlate the state of a page with respect to logged updates of that page. All updates of a transaction are logged, including those performed during rollbacks. By appropriate chaining of the log records written during rollbacks to those written during forward progress, a bounded amount of logging is ensured during rollbacks even in the face of repeated failures during restart or of nested rollbacks We deal with a variety of features that are very important in building and operating an *industrial-strength* transaction processing system ARIES supports fuzzy checkpoints, selective and deferred restart, fuzzy image copies, media recovery, and high concurrency lock modes (e.g., increment/decrement) which exploit the semantics of the operations and require the ability to perform operation logging. ARIES is flexible with respect to the kinds of buffer management policies that can be implemented. It supports objects of varying length efficiently. By enabling parallelism during restart, page-oriented redo, and logical undo, it enhances concurrency and performance. We show why some of the System R paradigms for logging and recovery, which were based on the shadow page technique, need to be changed in the context of WAL. We compare ARIES to the WAL-based recovery methods of

ACM Trans. Database Syst. 1992.

Agenda

Durability

Write ahead logging

- Force vs. No Force
- Steal vs. No Steal

ARIES logging

Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

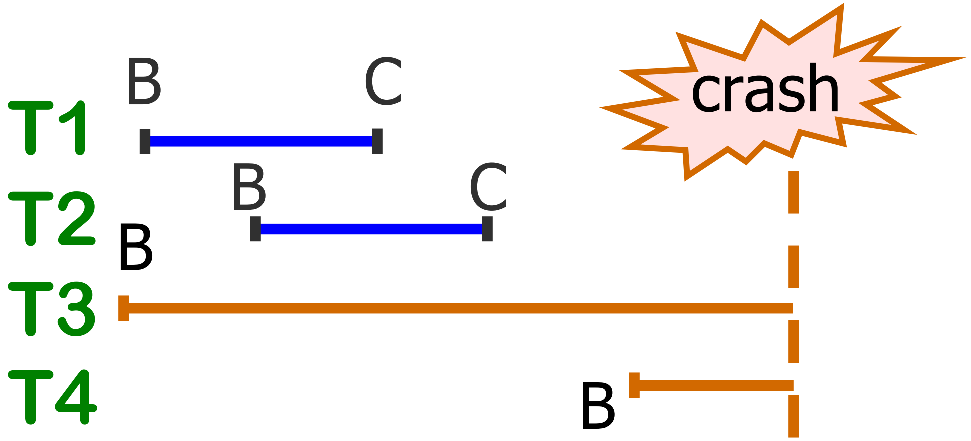
Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

Desired Behavior after system restarts

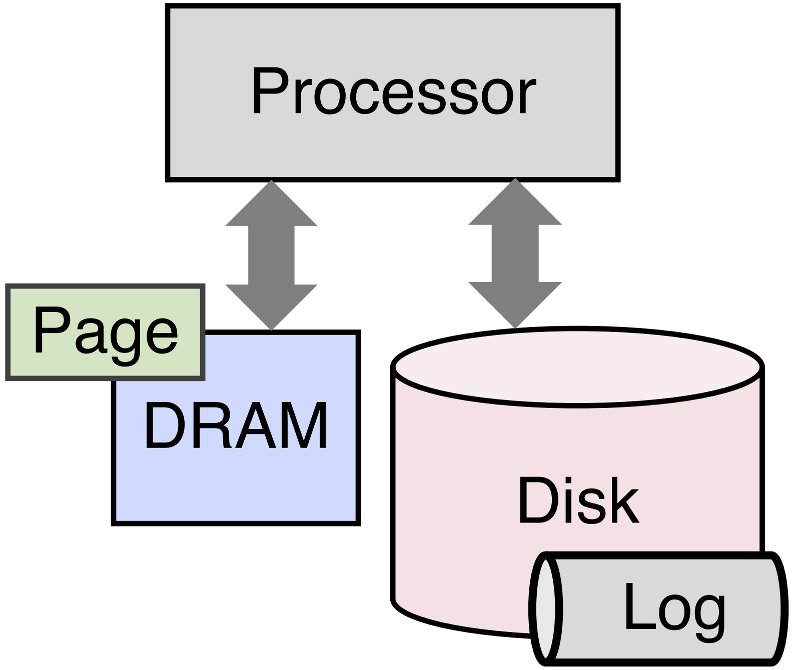
- T1, T2 should persist
- T3, T4 should be aborted



Write-Ahead Logging (WAL)

Before a transaction commits, its modifications must persist

Before writing dirty data to disk, rollback information must persist



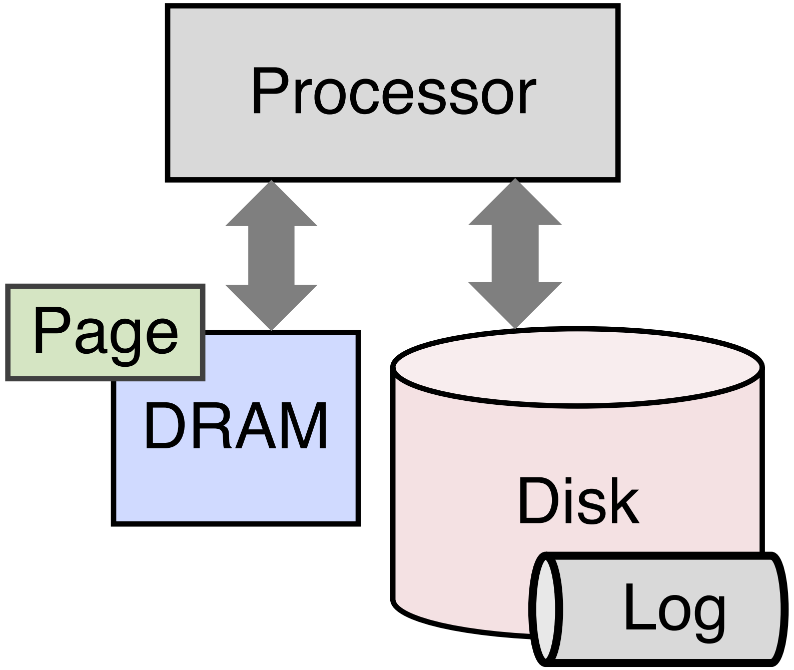
Write-Ahead Logging (WAL)

Before a transaction commits, its modifications must persist

Before writing dirty data to disk, rollback information must persist

Write-ahead logging: changes are written to the log before updating the database tables

– Writing to log incurs sequential IO



Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Steal: Dirty pages can be flushed to disk before the transaction commits

- Advantage: other transactions can use the buffer slot in DRAM
- Challenge: system crashes after flushing dirty pages but before the transaction commits
  => Dirty data on disk
- Solution: **UNDO logging** before each update

Buffer Management Policy

Force: All dirty pages must be flushed when the transaction commits

Buffer Management Policy

Force: All dirty pages must be flushed when the transaction commits

No Force: Dirty pages may stay in memory after the transaction commits

- Advantage: reduce # random IO
- Challenge: system crashes after the transaction commits but before the dirty pages are flushed
  => missing updates from committed transactions
- Solution: **REDO logging** before each update

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Disk-based DB

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Disk-based DB **Main memory DB**

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Non-volatile memory DB

Disk-based DB

Main memory DB

Baseline REDO/UNDO Design

Write: Write REDO/UNDO to log; update the page

Commit: Write COMMIT to log

Recovery:

- Forward scan of entire log: redo all records
- Backward scan of entire log: undo uncommitted transactions

Data structures

Log entry
- (LSN), txnID, pageID, data

Data page
- Tuple data

Baseline REDO/UNDO Design

Write: Write REDO/UNDO to log; update the page

Commit: Write COMMIT to log

Recovery:

- Forward scan of entire log: redo all records; **keep a table for active transactions**
- Backward scan of entire log: undo uncommitted transactions

Data structures

Log entry

- (LSN), txnID, pageID, data

Data page

- Tuple data

(Active) **Transaction Table**

- TransID

Limitation of the Baseline Design

Inefficiency in the REDO process

- Unnecessary to redo all records
- Need to redo only records that are not reflected in data pages

Limitation of the Baseline Design

Inefficiency in the REDO process

- Unnecessary to redo all records
- Need to redo only records that are not reflected in data pages

Inefficiency in the UNDO process

- Unnecessary to scan the entire log
- Need to undo only records of uncommitted transactions

Limitation of the Baseline Design

Inefficiency in the REDO process

- Unnecessary to redo all records
- Need to redo only records that are not reflected in data pages

Inefficiency in the UNDO process

- Unnecessary to scan the entire log
- Need to undo only records of uncommitted transactions

Lack of checkpointing

- Unnecessary to start from the beginning of log
- Start with the first log record that is not reflected in data pages

Optimize REDO Process

Inefficiency in the REDO process

- Unnecessary to redo all records
- Need to redo only records that are not reflected in the data page

Data structures

<u>Log entry</u>
- (LSN), txnID, pageID, data

<u>Data page</u>
- Tuple data

<u>(Active) Transaction Table</u>
- TransID

Optimize REDO Process

Inefficiency in the REDO process

- Unnecessary to redo all records
- Need to redo only records that are not reflected in the data page

Solution: add a version number to each page

- **pageLSN**: LSN of the log record that describes the latest update to the page.
- **REDO scan**: Apply REDO only if record.LSN > page.pageLSN
- **Write**: update pageLSN (for the buffered page) for each write

Data structures

Log entry
- (LSN), txnID, pageID, data

Data page
- Tuple data
- **pageLSN**

(Active) Transaction Table
- TransID

Optimize UNDO Process

Inefficiency in the UNDO process

- Unnecessary to scan the entire log
- Need to undo only records of uncommitted transactions

Data structures

Log entry

- (LSN), txnID, pageID, data

Data page

- tuple data
- pageLSN

(Active) Transaction Table

- transID

Optimize UNDO Process

Inefficiency in the UNDO process

- Unnecessary to scan the entire log
- Need to undo only records of uncommitted transactions

Solution: link records from the same transaction

- **prevLSN**: preceding log record written by the same transaction
- **lastLSN**: LSN of the last log record written by the transaction
- **UNDO scan**: Follow lastLSN and prevLSN to undo records
- **REDO scan**: update lastLSN in Transaction Table based on the last update of the transaction

Data structures

Log entry

- (LSN), txnID, pageID, data
- **prevLSN**

Data page

- tuple data
- pageLSN

(Active) Transaction Table

- transID
- **lastLSN**

Checkpoint

Lack of checkpointing

- Unnecessary to start from the beginning of log
- Start with the first log record that is not reflected in data pages

Data structures

Log entry
- (LSN), txnID, pageID, data
- prevLSN

Data page
- tuple data
- pageLSN

(Active) Transaction Table
- transID
- lastLSN

Checkpoint

Lack of checkpointing

- Unnecessary to start from the beginning of log
- Start with the first log record that is not reflected in data pages

Solution: Maintain a dirty page table

- **pageID**: ID of the dirty page
- **recLSN**: LSN of the first log record since when the page is dirty
- **Fuzzy Checkpoint:** log DPT and TT asynchronously
- **REDO scan**: start from the smallest LSN in DP

Data structures

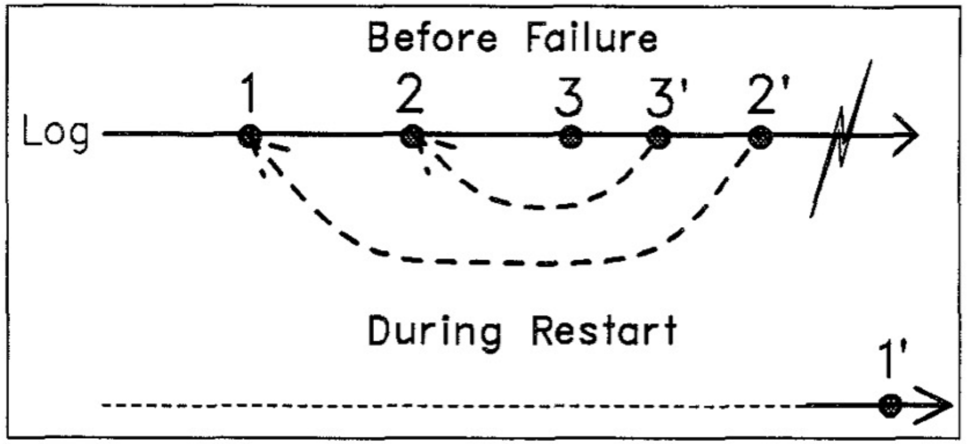
Log entry
- (LSN), txnID, pageID, data
- prevLSN

Data page
- tuple data
- pageLSN

(Active) Transaction Table
- transID
- lastLSN

Dirty Page Table
- **pageID**
- **recLSN**

Compensation Log Record (CLR)



I′ is the Compensation Log Record for I
I′ points to the predecessor, if any, of I

The action of applying UNDO leads to a CLR

- In undo scan, do not reapply UNDO if CLR exists
- **UndoNxtLSN**: LSN of the next record to be processed during undo scan

Data structures

Log entry

- (LSN), txnID, pageID, data
- prevLSN
- **UndoNxtLSN**

Data page

- tuple data
- pageLSN

(Active) Transaction Table

- transID
- lastLSN
- **UndoNxtLSN**

Dirty Page Table

- pageID
- recLSN

ARIES – Big Picture

Goal: Bring the database to the state before the crash (REDO phase) and rollback uncommitted transactions (UNDO phase)

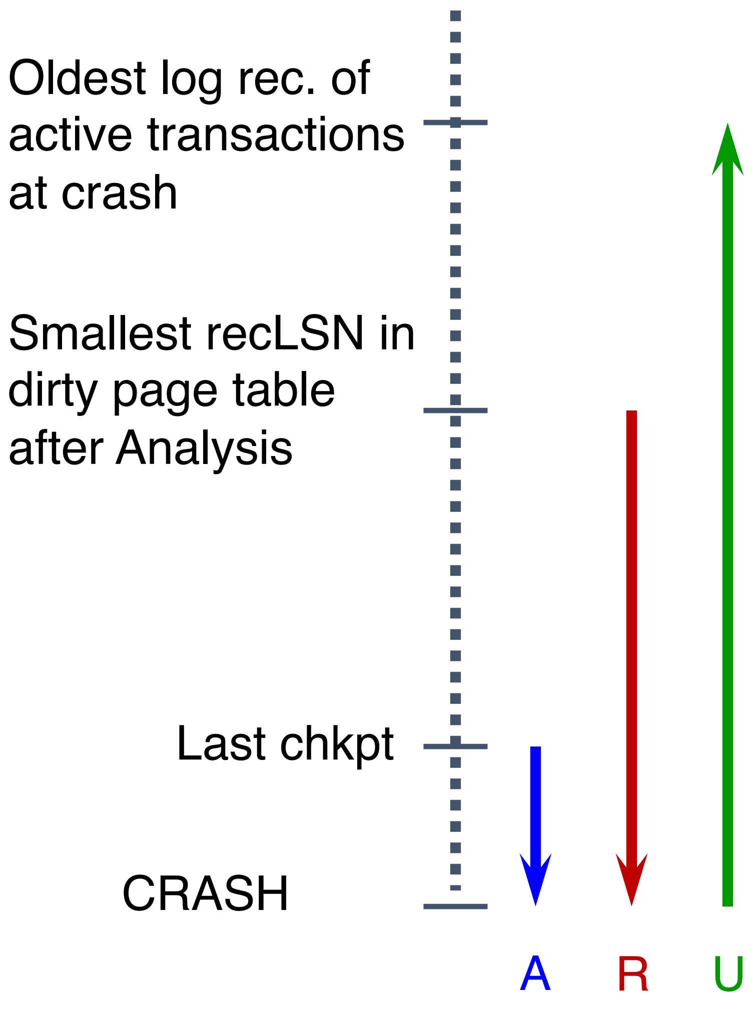
ARIES – Big Picture

Goal: Bring the database to the state before the crash (REDO phase) and rollback uncommitted transactions (UNDO phase)

Start from the last complete checkpoint

- **Analysis phase**: rebuild transaction table (for undo phase) and dirty page table (for redo phase)
- **REDO phase**: redo transactions whose effects may not be persistent before the crash
- **UNDO phase**: undo transactions that did not commit before the crash

ARIES – Big Picture



Goal: Bring the database to the state before the crash (REDO phase) and rollback uncommitted transactions (UNDO phase)

Start from the last complete checkpoint

- **Analysis phase**: rebuild transaction table (for undo phase) and dirty page table (for redo phase)
- **REDO phase**: redo transactions whose effects may not be persistent before the crash
- **UNDO phase**: undo transactions that did not commit before the crash

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

(update transaction table) For each log record:

- If ‘update’ or ‘CLR’: insert to transaction table if not exists
- If ‘end’: delete from transaction table

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

(update transaction table) For each log record:

- If ‘update’ or ‘CLR’: insert to transaction table if not exists
- If ‘end’: delete from transaction table

(update dirty page table) For each log record:

- If ‘update’ or ‘CLR’: insert to dirty page table if not exists (PageID, RecLSN)

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN

Dirty page table

PageID	RecLSN

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	**update: T1 writes P5**
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T1	**10**

Dirty page table

PageID	RecLSN
P5	**10**

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	**update: T2 writes P3**
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T1	10
T2	**20**

Dirty page table

PageID	RecLSN
P5	10
P3	**20**

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	**T1 abort**
40	**CLR: Undo T1 LSN 10**
45	**T1 End**
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
~~T1~~	~~10~~
T2	20

Dirty page table

PageID	RecLSN
P5	10
P3	20

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	**update: T3 writes P1**
60	update: T2 writes P5
X	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	**50**
T2	20

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	**50**

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	**update: T2 writes P5**
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	**60**

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Where to start?

- From log record containing **smallest RecLSN** in the dirty page table
- Before this LSN, all redo records have been reflected in data pages on disk

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Where to start?

- From log record containing **smallest RecLSN** in the dirty page table
- Before this LSN, all redo records have been reflected in data pages on disk

Observation: can **skip a redo record** for the following cases where the corresponding page has already been flushed before the crash

- The page is not in dirty page table (DPT)
- The page is in DPT but redo_record.LSN < DPT[page].recLSN
- After fetching the data page, redo_record.LSN ≤ page.page_LSN

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	**update: T1 writes P5**
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

No need to update

Write already reflected on disk

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	**update: T2 writes P3**
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

Update P3 in buffer pool

No need to flush P3 now

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	**CLR: Undo T1 LSN 10**
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

No need to update

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

Write already reflected on disk

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	**update: T3 writes P1**
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

Update P1 in buffer pool

No need to flush P1 now

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	**update: T2 writes P5**
X	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

Update P5 in buffer pool

No need to flush P5 now

Crash Recovery – UNDO Phase

Rollback uncommitted transactions

Crash Recovery – UNDO Phase

Rollback uncommitted transactions

Repeat until transaction table is empty:

- Choose largest LastLSN among transactions in the transaction table
- If the log record is an 'update': Undo the update, write a CLR, add record.prevLSN to transaction table
- If the log record is an 'CLR': add CLR.UndoNxtLSN to transaction table
- If prevLSN and UpdoNxtLSN are NULL, remove the transaction from transaction table

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	50	50
T2	60	60

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	**update: T2 writes P5**
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	50	50
T2	~~60~~ 70	~~60~~ 20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	**update: T3 writes P1**
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	~~50~~ 80	~~50~~ null
T2	70	20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
X	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
~~T3~~	~~80~~	~~null~~
T2	70	20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	**update T2 writes P3**
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T2	~~70~~ 90	~~20~~ null

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	
90	CLR: Undo T2, LSN 20,	(null)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
~~T2~~	~~90~~	~~null~~

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	
90	CLR: Undo T2, LSN 20,	(null)
95	T2 End	

Crash During Restart – Example

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: Undo T1 LSN 10, T1 End
50	update: T3 writes P1
60	update: T2 writes P5
✕	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
✕	CRASH, RESTART
90	CLR: Undo T2 LSN 20, T2 end

No need to undo LSN 60 and LSN 50 again due to the CLRs created in the previous restart

Can created a checkpoint to reduce the cost of future restart

Q/A – ARIES

Alternatives to logging to provide atomicity and durability?

Logging and recovery in modern database systems?

Challenges in ARIES for modern memory hierarchy?

ARIES with disaggregated storage?

How does the buffer manager obey write ahead logging?

Does the redo order matter?

How much does time/space overhead of logging affect the system?

Next Week

Next Monday: Exam review

- Lecture given by TA
- Exam questions in previous years available on course website

Next Wednesday–Friday: Take-home exam