Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)
  - Desired state after system restarts:
    - T1 & T3 should be durable.
    - T2, T4 & T5 should be aborted (effects not seen).

General Overview

- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
Main ideas so far:

- Write-Ahead Log, for loss of volatile storage,
- with incremental updates (STEAL, NO FORCE)
- and checkpoints
- On recovery: **undo** uncommitted; **redo** committed transactions.

Today: ARIES

With full details on
- fuzzy checkpoints
- recovery algorithm

C. Mohan (IBM)

ARIES – guiding principles

1) **Write ahead**
2) **Fast**, during normal operation
   A. Least interference with OS (‘STEAL’, ‘NO FORCE’)
   B. Fast (fuzzy) checkpoints
3) On multi-step ops: `<begin op> … <end op>`
4) **Redo everything**: undo ‘loser’ xacts
Overview

• Preliminaries
• Write-Ahead Log - main ideas
• (Shadow paging)
• Write-Ahead Log: ARIES
  – LSN’s
  – examples of normal operation & of abort
  – fuzzy checkpoints
  – recovery algo

LSN etc

<table>
<thead>
<tr>
<th>Name</th>
<th>where</th>
<th>dfn</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN</td>
<td></td>
<td>Log seq. #</td>
</tr>
<tr>
<td>flushedLSN</td>
<td>RAM</td>
<td>Last LSN on log</td>
</tr>
<tr>
<td>pageLSN</td>
<td>@page_i</td>
<td>Latest update to page_i</td>
</tr>
<tr>
<td>recLSN</td>
<td>@page_i</td>
<td>Earliest update “ ”</td>
</tr>
<tr>
<td>lastLSN</td>
<td>T_j</td>
<td>Latest action of T_j</td>
</tr>
<tr>
<td>Master record</td>
<td></td>
<td>LSN of latest checkpoint</td>
</tr>
</tbody>
</table>

LSN

• Log Sequence Number
• every log record has an LSN
• Q: Why do we need it?
Types of log records

Q1: Which types?
A1: Update, commit, checkpoint, …
Q2: What format?
A2: x-id, type, (old value, …)
Log Records

Possible log record types:
- Update, Commit, Abort
- Checkpoint (for log maintenance)
- Compensation Log Records (CLRs)
  - for UNDO actions
- End (end of commit or abort)

LogRecord fields:
- prevLSN
- XID
- type
- pageID
- length
- offset
- before-image
- after-image

Overview

- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
  - LSN’s
  - examples of normal operation & of abort
  - fuzzy checkpoints
  - recovery algo

Writing log records

- We don’t want to write one record at a time
  - (why not?)
- How should we buffer them?
Writing log records

• We don’t want to write one record at a time
  – (why not?)
• How should we buffer them?
  – Batch log updates;
  – Un-pin a data page ONLY if all the corresponding log records have been flushed to the log.

WAL & the Log

• Each data page contains a pageLSN.
  – The LSN of the most recent update to that page.
• System keeps track of flushedLSN.
  – The max LSN flushed so far.
• WAL: For a page \(i\) to be written must flush log at least to the point where:
  \[ \text{pageLSN}_i \leq \text{flushedLSN} \]

WAL & the Log

• Can we un-pin the gray page?
WAL & the Log

- Can we un-pin the gray page?
- A: yes

WAL & the Log

- Can we un-pin the red page?

WAL & the Log

- Can we un-pin the red page?
- A: no
WAL & the Log

- LSNs
- DB pageLSNs
- RAM flushedLSN

Q: why not on disk or log?

Overview

- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
  - LSN’s
  - examples of normal operation & of abort
  - fuzzy checkpoints
  - recovery algo

Normal Execution of an Xact

- Series of reads & writes, followed by commit or abort.
  - We will assume that disk write is atomic.
    - In practice, additional details to deal with non-atomic writes.
  - Strict 2PL.
  - STEAL, NO-FORCE buffer management, with Write-Ahead Logging.
Normal execution of an Xact

- Page ‘i’ can be written out only after the corresponding log record has been flushed

Transaction Commit

- Write commit record to log.
- All log records up to Xact’s commit record are flushed to disk.

Q: why not flush the dirty pages, too?

A: speed – ‘NO FORCE’
Transaction Commit

• Write **commit record to log**.
• All log records up to Xact’s **commit record** are flushed to disk.
  – Note that log flushes are sequential, synchronous writes to disk.
  – Many log records per log page.
• Commit() returns.
• Write **end record to log**.
Overview

- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
  - LSN’s
  - examples of normal operation & of **abort**
  - fuzzy checkpoints
  - recovery algo

Abort

Actually, a special case of the up-coming ‘undo’ operation, applied to only one transaction - e.g.:

Abort - Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN tid type</th>
<th>item</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>NULL T2 update</td>
<td>Y</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>10 T2 abort</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Abort - Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>tid</th>
<th>type</th>
<th>item</th>
<th>old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>NULL</td>
<td>T2</td>
<td>update</td>
<td>Y</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>63</td>
<td>10</td>
<td>T2</td>
<td>abort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>63</td>
<td>T2</td>
<td>CLR</td>
<td>Y</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>78</td>
<td>72</td>
<td>T2</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compensating log record

## CLR record - details

- A CLR record has all the fields of an 'update' record
- Plus the 'undoNextLSN' pointer, to the next-to-be-undone LSN
Abort - algorithm:

- First, write an ‘abort’ record on log and
- Play back updates, in reverse order: for each update
  - write a CLR log record
  - restore old value
- at end, write an ‘end’ log record

Notice: CLR records never need to be undone

Overview

- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
  - LSN’s
  - examples of normal operation & of abort
    - fuzzy checkpoints
    - recovery algo

(non-fuzzy) checkpoints

- they have performance problems - recall from previous lecture:
(non-fuzzy) checkpoints

We assumed that the DBMS:
• stops all transactions, and
• flushes on disk the ‘dirty pages’
Both decisions are expensive
Q: Solution?

Hint 1: record state as of the beginning of the ckpt
Hint 2: we need some guarantee about which pages made it to the disk

Q: Solution?
A: write on the log:
• the id-s of active transactions and
• the id-s (ONLY!) of dirty pages (rest: obviously made it to the disk!)
Q: Solution?
A: write on the log:
- the id-s of active transactions and
- the id-s (ONLY!) of dirty pages (rest: obviously made it to the disk!)

(Fuzzy) checkpoints

Specifically, write to log:
- `begin_checkpoint` record: indicates start of ckpt
- `end_checkpoint` record: Contains current `Xact table` and `dirty page table`. This is a `fuzzy checkpoint`:
  - Other Xacts continue to run; so these tables accurate only as of the time of the `begin_checkpoint` record.
  - No attempt to force dirty pages to disk; effectiveness of checkpoint limited by oldest unwritten change to a dirty page.

solved both problems of non-fuzzy ckpts!!
(Fuzzy) checkpoints - cont’d

And:

– Store LSN of most recent chkpt record on disk (master record)
  • Q: why do we need that?

Q: why do we need that?

• A: so that we know where to start from, on crash & recovery

(Fuzzy) Checkpoints

More details: Two in-memory tables:

#1) Transaction Table

Q: what would you store there?
(Fuzzy) Checkpoints
More details: Two in-memory tables:

#1) Transaction Table
- One entry per currently active Xact.
  - entry removed when Xact commits or aborts
- Contains
  - XID,
  - status (running/committing/aborting), and
  - lastLSN (most recent LSN written by Xact).

#2) Dirty Page Table:
- One entry per dirty page currently in buffer pool.
- Contains recLSN -- the LSN of the log record which first caused the page to be dirty.

Overview
- Preliminaries
- Write-Ahead Log - main ideas
- (Shadow paging)
- Write-Ahead Log: ARIES
  - LSN’s
  - examples of normal operation & of abort
  - fuzzy checkpoints
  - recovery algo
The Big Picture: What’s Stored Where

- **DB**
  - Data pages: each with a pageLSN
- **RAM**
  - Xact Table: lastLSN, status
  - Dirty Page Table: recLSN, flushedLSN
  - LOG:
    - LogRecords: prevLSN, XID, type, length, pageID, offset, before-image, after-image
    - master record: LSN of most recent checkpoint

Crash Recovery: Big Picture

- Start from a checkpoint (found via master record).
- Three phases:
  - Analysis: Figure out which Xacts committed since checkpoint, which failed.
  - REDO all actions (repeat history)
  - UNDO effects of failed Xacts.

Notice: relative ordering of A, B, C may vary!
Recovery: The Analysis Phase

• Re-establish knowledge of state at checkpoint.
  – via transaction table and dirty page table stored in the checkpoint

<end ckpt, [T499], {P10,P12}>

Recovery: The Analysis Phase

• Scan log forward from checkpoint.
  – End record: Remove Xact from Xact table.
  – All Other records:
    • Add Xact to Xact table, with status ‘U’ (=candidate for undo)
    • set lastLSN=LSN,
    • on commit, change Xact status to ‘C’.
  – also, for Update records: If page P not in Dirty Page Table,
    • add P to DPT, set its recLSN=LSN.

Recovery: The Analysis Phase

• At end of Analysis:
  – transaction table says which xacts were active at time of crash.
  – DPT says which dirty pages might not have made it to disk
Recovery: The Analysis Phase

Example

<table>
<thead>
<tr>
<th>Log Entry</th>
<th>Transaction</th>
<th>Dirty Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN 10 &lt;begin ckpt&gt;</td>
<td>(T96, U)</td>
<td>(P33)</td>
</tr>
<tr>
<td>LSN 20 &lt;T96 P33, A, 10, 15&gt;</td>
<td>(T96,U), (T33,U)</td>
<td>(P33), (P20)</td>
</tr>
<tr>
<td>LSN 30 &lt;end ckpt {T96, T33}, {P20, P33}&gt;</td>
<td>(T96,C), (T33,U)</td>
<td>(P33), (P20)</td>
</tr>
<tr>
<td>LSN 40 &lt;T96 commit&gt;</td>
<td>(T33,U)</td>
<td>(P33), (P20)</td>
</tr>
</tbody>
</table>

Phase 2: REDO

Goal: repeat History to reconstruct state at crash:

- Reapply all updates (even of aborted Xacts!), redo CLRs.
- (and try to avoid unnecessary reads and writes!)

Specifically:

- Scan forward from log rec containing smallest recLSN in DPT.  Q: why start here?

Phase 2: REDO

Goal: repeat History to reconstruct state at crash:

- Reapply all updates (even of aborted Xacts!), redo CLRs.
- (and try to avoid unnecessary reads and writes!)

Specifically:

- Scan forward from log rec containing smallest recLSN in DPT.  Q: why start here?

A: all else have been flushed
Crash Recovery: Big Picture

- Start from a checkpoint (found via master record).
- Three phases:
  - Analysis - Figure out which Xacts committed since checkpoint, which failed.
  - REDO all actions (repeat history)
  - UNDO effects of failed Xacts.

Phase 2: REDO (cont’d)

- For each update log record or CLR with a given LSN, REDO the action unless:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN. (this last case requires I/O)

Phase 2: REDO (cont’d)

- To REDO an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging, no forcing!
Phase 2: REDO (cont’d)

- ...
- at the end of REDO phase, write ‘end’ log records for all xacts with status ‘C’,
- and remove them from transaction table

Phase 3: UNDO

Goal: Undo all transactions that were active at the time of crash (‘loser xacts’)

- That is, all xacts with ‘U’ status on the xact table of the Analysis phase
- Process them in reverse LSN order
- using the lastLSN’s to speed up traversal
- and issuing CLR's

Phase 3: UNDO

Goal: Undo all transactions that were active at the time of crash (‘loser xacts’)

\[
\begin{array}{ll}
\text{Dirty Page} & \text{xact-table} \\
\text{table} & \\
(P33) & (T96, U) \\
(P33),(P20) & (T96,U),(T33,U) \\
(P33),(P20) & (T96,C),(T33,U) \\
(P33),(P20) & (T33,U) \\
\end{array}
\]
Phase 3: UNDO

- \( \text{ToUndo} = \{ \text{lastLSNs of `loser` Xacts} \} \)

**Repeat:**

- Choose (and remove) largest LSN among \( \text{ToUndo} \).
- If this LSN is a CLR and \( \text{undonextLSN} = \text{NULL} \)
  - Write an End record for this Xact.
- If this LSN is a CLR, and \( \text{undonextLSN} \neq \text{NULL} \)
  - Add \( \text{undonextLSN} \) to \( \text{ToUndo} \)
- Else this LSN is an update. Undo the update, write a CLR, add \( \text{prevLSN} \) to \( \text{ToUndo} \).

**Until \( \text{ToUndo} \) is empty.**

---

**Phase 3: UNDO - illustration**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Suppose that after end of analysis phase we have:

- xact table
- xid   status   lastLSN
- T32   U          
- T41   U          

---

**Phase 3: UNDO - illustration**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Suppose that after end of analysis phase we have:

- xact table
- xid   status   lastLSN
- T32   U          
- T41   U          

Undo in reverse LSN order.
Example of Recovery

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>begin_checkpoint</td>
</tr>
<tr>
<td>05</td>
<td>end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40</td>
<td>CLR: Undo T1, LSN 10</td>
</tr>
<tr>
<td>45</td>
<td>T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
</tbody>
</table>

Example of Recovery Diagram:
- RAM
- Xact Table
  - lastLSN
  - status
- Dirty Page Table
  - recLSN
  - flushedLSN
- ToUndo

Example of Recovery Table:
- lastLSN
- status
- Dirty Page Table
  - recLSN
  - flushedLSN
  - ToUndo
  - prevLSNs

Questions

- Q1: After the Analysis phase, which are the ‘loser’ transactions?
  - T2 and T3
- Q2: UNDO phase - what will it do?
  - undo ops of LSN 60, 50, 20
Example: Crash During Restart!

begin_checkpoint, end_checkpoint
update: T1 writes P5
update T2 writes P3
30  T1 abort
CLR: undo T1 LSN 10, T1 End
50  update T3 writes P1
60  update T3 writes P5
CRASH, RESTART

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update T3 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
</tbody>
</table>

CRASH, RESTART

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update T3 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: undo T2 LSN 60'</td>
</tr>
</tbody>
</table>

CRASH, RESTART

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update T3 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: undo T3 LSN 50, T3 end</td>
</tr>
</tbody>
</table>
Example: Crash During Restart!

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P2</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: Undo T2 LSN 60</td>
</tr>
<tr>
<td>90</td>
<td>CLR: Undo T3 LSN 50, T3 end</td>
</tr>
<tr>
<td>100</td>
<td>CRASH, RESTART</td>
</tr>
</tbody>
</table>

Questions

• Q3: After the Analysis phase, which are the ‘loser’ transactions?
  • A3: T2 only

• Q4: UNDO phase - what will it do?
  • A4: follow the string of prevLSN of T2, exploiting undoNextLSN
Example: Crash During Restart!

- Q5: show the log, after the recovery is finished:
Additional Crash Issues

- What happens if system crashes during Analysis? During REDO?
- How do you limit the amount of work in REDO?
  - Flush asynchronously in the background.
- How do you limit the amount of work in UNDO?
  - Avoid long-running Xacts.

Summary of Logging/Recovery

- Recovery Manager guarantees Atomicity & Durability.

  Atomicity
  Consistency
  Isolation
  Durability

Summary of Logging/Recovery

ARIES - main ideas:
- WAL (write ahead log), STEAL/NO-FORCE
- fuzzy checkpoints (snapshot of dirty page ids)
- redo everything since the earliest dirty page; undo ‘loser’ transactions
- write CLRs when undoing, to survive failures during restarts
Summary of Logging/Recovery

Additional concepts:
• LSNs identify log records; linked into backwards chains per transaction (via prevLSN).
• pageLSN allows comparison of data page and log records.
• (and several other subtle concepts: undoNextLSN, recLSN etc)