

CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

Lecture #25: Crash Recovery - part 2
 (R&G, ch. 18)

CMU SCS

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).

T1 — Commit
 T2 — Abort
 T3 — Commit
 T4 — Commit
 T5 — Commit

crash!

Faloutsos CMU SCS 15-415/615 2

CMU SCS

General Overview

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

Faloutsos CMU SCS 15-415/615 3

CMU SCS

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.


Faloutsos CMU SCS 15-415/615 4

CMU SCS

Today: ARIES

With full details on

- fuzzy checkpoints
- recovery algorithm



C. Mohan (IBM)

Faloutsos CMU SCS 15-415/615 5

CMU SCS

Write on board

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

Faloutsos CMU SCS 15-415/615 6

CMU SCS

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

Faloutsos CMU SCS 15-415/615 7

CMU SCS

LSN etc

Write on board

Name	where	dfn
LSN		Log seq. #
flushedLSN	RAM	Last LSN on log
pageLSN	@page _i	Latest update to page _i
recLSN	@page _i	Earliest update “ “
lastLSN	T _j	Latest action of T _j
Master record		LSN of latest checkpoint

Faloutsos CMU SCS 15-415/615 8

CMU SCS

LSN

- Log Sequence Number
- every log record has an LSN
- Q: Why do we need it?

Faloutsos CMU SCS 15-415/615 9

CMU SCS

LSN

A1: e.g, undo T4 - it is faster, if we have a linked list of the T4 log records

A2: and many other uses - see later

```

<T1 start>
<T2 start>
<T4 start>
<T4, A, 10, 20>
<T1 commit>
<T4, B, 30, 40>
<T3 start>
<T2 commit>
<T3 commit>
~~~~ CRASH ~~~~

```

Faloutsos CMU SCS 15-415/615 10

CMU SCS

Types of log records

Q1: Which types?

A1:

Q2: What format?

A2:

```

<T1 start>
<T2 start>
<T4 start>
<T4, A, 10, 20>
<T1 commit>
<T4, B, 30, 40>
<T3 start>
<T2 commit>
<T3 commit>
~~~~ CRASH ~~~~

```

Faloutsos CMU SCS 15-415/615 11

CMU SCS

Types of log records

Q1: Which types?

A1: Update, commit, ckpoint, ...

Q2: What format?

A2: x-id, type, (old value, ...)

```

<T1 start>
<T2 start>
<T4 start>
<T4, A, 10, 20>
<T1 commit>
<T4, B, 30, 40>
<T3 start>
<T2 commit>
<T3 commit>
~~~~ CRASH ~~~~

```

Faloutsos CMU SCS 15-415/615 12

CMU SCS

Log Records

LogRecord fields:

- prevLSN
- XID
- type
- pageID
- length
- offset
- before-image
- after-image

update records only

Possible log record types:

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

Faloutsos CMU SCS 15-415/615 13

CMU SCS

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

Faloutsos CMU SCS 15-415/615 14

CMU SCS

Writing log records

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

Faloutsos CMU SCS 15-415/615 15

CMU SCS

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.

Faloutsos CMU SCS 15-415/615 16

CMU SCS

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:
 $pageLSN_i \leq flushedLSN$

Faloutsos CMU SCS 15-415/615 17

CMU SCS

WAL & the Log

- Can we un-pin the gray page?

Faloutsos CMU SCS 15-415/615 18

CMU SCS

WAL & the Log

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

Log records flushed to disk

flushedLSN

pageLSN "Log tail" in RAM

Faloutsos CMU SCS 15-415/615 19

CMU SCS

WAL & the Log

- Can we un-pin the red page?

Log records flushed to disk

flushedLSN

pageLSN "Log tail" in RAM

Faloutsos CMU SCS 15-415/615 20

CMU SCS

WAL & the Log

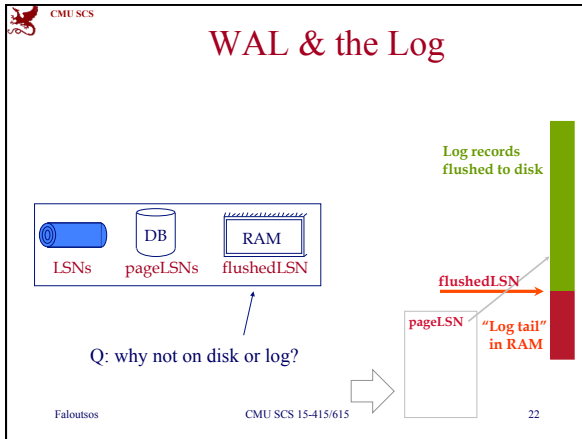
- Can we un-pin the red page?
- A: no

Log records flushed to disk

flushedLSN

pageLSN "Log tail" in RAM

Faloutsos CMU SCS 15-415/615 21



- ### 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.**

CMU SCS

Normal execution of an Xact

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

Faloutsos CMU SCS 15-415/615 25

CMU SCS

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?

Faloutsos CMU SCS 15-415/615 26

CMU SCS

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'

Faloutsos CMU SCS 15-415/615 27

CMU SCS

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.

Faloutsos CMU SCS 15-415/615 28

CMU SCS

Example

LSN	prevLSN	tid	type	item	old	new
10	NULL	T1	update	X	30	40
...						
50	10	T1	update	Y	22	25
...						
63	50	T1	commit			
...						
68	63	T1	end			

dbms flushes
log records
+ some
record-keeping

Faloutsos CMU SCS 15-415/615 29

CMU SCS

Example

LSN	prevLSN	tid	type	item	old	new
10	NULL	T1	update	X	30	40
...						
50	10	T1	update	Y	22	25
...						
63	50	T1	commit			
...						
68	63	T1	end			

dbms flushes
log records
+ some
record-keeping

Faloutsos CMU SCS 15-415/615 30

CMU SCS

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

Faloutsos CMU SCS 15-415/615 31

CMU SCS

Abort

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

Faloutsos CMU SCS 15-415/615 32

CMU SCS

Abort - Example

LSN	<i>prevLSN</i>	<i>tid</i>	<i>type</i>	<i>item</i>	<i>old</i>	<i>new</i>
10	NULL	T2	update	Y	30	40
...						
63	10	T2	abort			

Faloutsos CMU SCS 15-415/615 33

CMU SCS

Abort - Example

LSN	prevLSN	tid	type	item	old	new
10	NULL	T2	update	Y	30	40
...						
63	10	T2	abort			
...						
72	63	T2	CLR (LSN 10)			
...						
78	72	T2	end			

} compensating log record

Faloutsos CMU SCS 15-415/615 34

CMU SCS

Abort - Example

LSN	prevLSN	tid	type	item	old	new	undoNextLSN
10	NULL	T2	update	Y	30	40	
...							
63	10	T2	abort				
...							
72	63	T2	CLR	Y	40	30	NULL
...							
78	72	T2	end				

Faloutsos CMU SCS 15-415/615 35

CMU SCS

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

Faloutsos CMU SCS 15-415/615 36

CMU SCS

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

Faloutsos CMU SCS 15-415/615 37

CMU SCS

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

Faloutsos CMU SCS 15-415/615 38

CMU SCS

(non-fuzzy) checkpoints

- they have performance problems - recall from previous lecture:

Faloutsos CMU SCS 15-415/615 39

CMU SCS reminder

(non-fuzzy) checkpoints

We assumed that the DBMS:

- stops all transactions, and
- flushes on disk the 'dirty pages'

Both decisions are expensive
Q: Solution?

```

<T1 start>
...
<T1 commit>
...
<T499, C, 1000, 1200>
<checkpoint>
<T499 commit>
<T500 start> before
<T500, A, 200, 400>
<checkpoint>
<T500, B, 10, 12>
crash
  
```

Faloutsos CMU SCS 15-415/615 40

CMU SCS reminder

(non-fuzzy) checkpoints

Q: Solution?

Hint1: record state as of the beginning of the ckpt

Hint2: we need some guarantee about which pages made it to the disk

```

<T1 start>
...
<T1 commit>
...
<T499, C, 1000, 1200>
<checkpoint>
<T499 commit>
<T500 start>
<T500, A, 200, 400>
<checkpoint>
<T500, B, 10, 12>
crash
  
```

Faloutsos CMU SCS 15-415/615 41

CMU SCS

checkpoints

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!)

```

<T1 start>
...
<T1 commit>
...
<T499, C, 1000, 1200>
<checkpoint, {T499}, {P10,P12}>
<T499 commit>
<T500 start>
<T500, A, 200, 400>
<checkpoint {T500}, {P10,P33}>
<T500, B, 10, 12>
crash
  
```

Faloutsos CMU SCS 15-415/615 42

CMU SCS

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!)

```

<T1 start>
...
<T1 commit>
...
<T499, C, 1000, 1200>
<begin ckpt>
...
<end ckpt, {T499}, {P10,P12}>
<T499 commit>
<T500 start>
<T500, A, 200, 400>
<begin ckpt>
...
<checkpoint {T500}, {P10,P33}>
<T500, B, 10, 12> crash

```

Faloutsos CMU SCS 15-415/615 43

CMU SCS

(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.

Faloutsos CMU SCS 15-415/615 44

CMU SCS

(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!!

Faloutsos CMU SCS 15-415/615 45

CMU SCS

(Fuzzy) checkpoints - cont'd

And:

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

Faloutsos CMU SCS 15-415/615 46

CMU SCS

(Fuzzy) checkpoints - cont'd

And:

- Store LSN of most recent chkpt record on disk (master record)
 - Q: why do we need that?
 - A: so that we know where to start from, on crash & recovery

Faloutsos CMU SCS 15-415/615 47

CMU SCS

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

(Fuzzy) Checkpoints

More details: Two in-memory tables:

#1) Transaction Table

Q: what would you store there?

Faloutsos CMU SCS 15-415/615 48

CMU SCS <end ckpt, {T499}, {P10,P12}>

(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).

Faloutsos CMU SCS 15-415/615 49

CMU SCS <end ckpt, {T499}, {P10,P12}>

(Fuzzy) Checkpoints

#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.

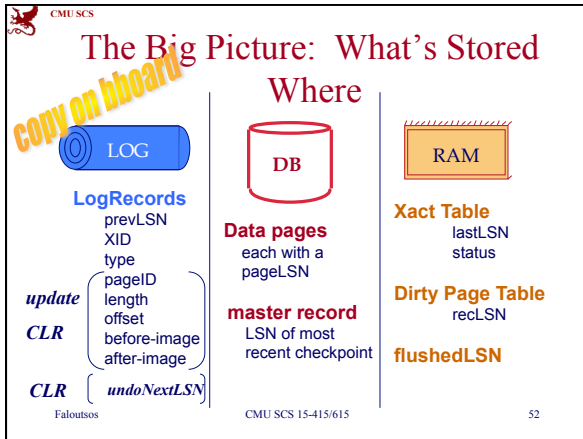
Faloutsos CMU SCS 15-415/615 50

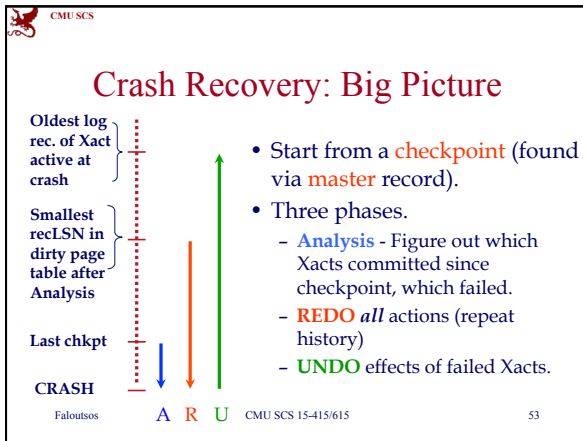
CMU SCS

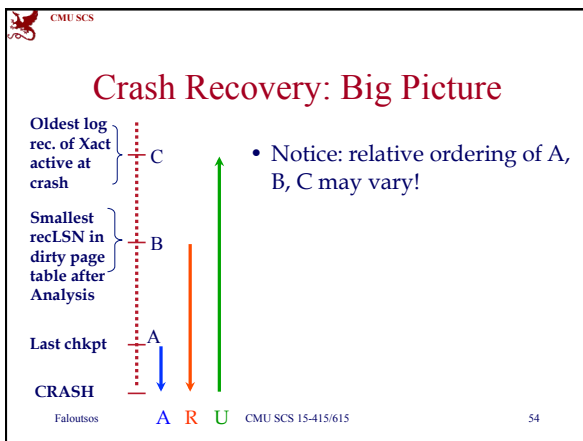
Overview

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

Faloutsos CMU SCS 15-415/615 51







CMU SCS

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}>

Faloutsos CMU SCS 15-415/615 55

CMU SCS

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**.

Faloutsos CMU SCS 15-415/615 56

CMU SCS

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

Faloutsos CMU SCS 15-415/615 57

CMU SCS

Recovery: The Analysis Phase

Example

...

LSN 10 <begin ckpt >

LSN 20 <T96 P33, A, 10, 15>

LSN 30 <end ckpt {T96, T33}, {P20, P33}>

LSN 40 <T96 commit>

LSN 50 <T96 end>

xact-table	Dirty Page table
(T96, U)	(P33)
(T96,U), (T33,U)	(P33), (P20)
(T96,C), (T33,U)	(P33), (P20)
(T33,U)	(P33), (P20)

crash

Faloutsos CMU SCS 15-415/615 58

CMU SCS

Phase 2: REDO

Goal: *repeat History* to reconstruct state at crash:

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

Specifically:

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

Faloutsos CMU SCS 15-415/615 59

CMU SCS

Phase 2: REDO

Goal: *repeat History* to reconstruct state at crash:

- Reapply *all* updates (even of aborted Xacts!), redo CLR's.
- (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

Faloutsos CMU SCS 15-415/615 60

CMU SCS reminder

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.

Faloutsos A R U CMU SCS 15-415/615 61

CMU SCS

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) \geq **LSN**. (this last case requires I/O)

Faloutsos A R CMU SCS 15-415/615 62

CMU SCS

Phase 2: REDO (cont'd)

- ...
- To **REDO** an action:
 - Reapply logged action.
 - Set **pageLSN** to **LSN**. No additional logging, no forcing!

Faloutsos CMU SCS 15-415/615 63

CMU SCS

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

Faloutsos CMU SCS 15-415/615 64

CMU SCS

Phase 3: UNDO

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

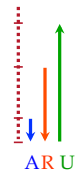
- That is, all xacts with 'U' status on the xact table of the Analysis phase
- Process them in reverse LSN order
- using the lastLSN's to speed up traversal
- and issuing CLR's

Faloutsos CMU SCS 15-415/615 65

CMU SCS

Phase 3: UNDO

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



xact-table	Dirty Page table
(T96, U)	(P33)
(T96,U), (T33,U)	(P33), (P20)
(T96,C), (T33,U)	(P33), (P20)
(T33,U)	(P33), (P20)

Faloutsos CMU SCS 15-415/615

CMU SCS

Phase 3: UNDO

ToUndo={lastLSNs of 'loser' Xacts}

Repeat:

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

Until ToUndo is empty.

Faloutsos CMU SCS 15-415/615 67

CMU SCS

Phase 3: UNDO - illustration

suppose that after end of analysis phase we have:

xact table

xid	status	lastLSN
T32	U	45
T41	U	50

Faloutsos CMU SCS 15-415/615 68

CMU SCS

Phase 3: UNDO - illustration

suppose that after end of analysis phase we have:

xact table

xid	status	lastLSN
T32	U	45
T41	U	50

Faloutsos CMU SCS 15-415/615 69

Example of Recovery

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
60	CRASH

RAM: Xact Table, Dirty Page Table, flushedLSN, lastLSN, status, recLSN, ToUndo

Faloutsos CMU SCS 15-415/615 70

Questions

- Q1: After the Analysis phase, which are the 'loser' transactions?
- Q2: UNDO phase - what will it do?

Faloutsos CMU SCS 15-415/615 71

Questions

- Q1: After the Analysis phase, which are the 'loser' transactions?
- A1: T2 and T3
- Q2: UNDO phase - what will it do?
- A2: undo ops of LSN 60, 50, 20

Faloutsos CMU SCS 15-415/615 72

Example: Crash During Restart!

RAM

Xact Table
lastLSN
status

Dirty Page Table
recLSN

flushedLSN

ToUndo

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

Example: Crash During Restart!

RAM

Xact Table
lastLSN
status

Dirty Page Table
recLSN

flushedLSN

ToUndo

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

undonextLSN

Example: Crash During Restart!

RAM

Xact Table
lastLSN
status

Dirty Page Table
recLSN

flushedLSN

ToUndo

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

undonextLSN

Example: Crash During Restart!

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

RAM

undonextLSN

Faloutsos CMU SCS 15-415/615 76

Questions

- Q3: After the Analysis phase, which are the 'loser' transactions?
- Q4: UNDO phase - what will it do?

Faloutsos CMU SCS 15-415/615 77

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*

Faloutsos CMU SCS 15-415/615 78

Example: Crash During Restart!

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

RAM

Xact Table
lastLSN status
Dirty Page Table
recLSN
flushedLSN

ToUndo

undonextLSN

Faloutsos CMU SCS 15-415/615 79

Questions

- Q5: show the log, after the recovery is finished:

Faloutsos CMU SCS 15-415/615 80

Example: Crash During Restart!

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, 95	CLR: Undo T2 LSN 20, T2 end

RAM

Xact Table
lastLSN status
Dirty Page Table
recLSN
flushedLSN

ToUndo

undonextLSN

Faloutsos CMU SCS 15-415/615 81

CMU SCS

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.

Faloutsos CMU SCS 15-415/615 82

CMU SCS

Summary of Logging/Recovery

- **Recovery Manager** guarantees Atomicity & Durability.

Atomicity
Consistency
Isolation
Durability

Faloutsos CMU SCS 15-415/615 83

CMU SCS

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

} let OS do its best

} idempotency

Faloutsos CMU SCS 15-415/615 84

 CMU SCS

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)

Faloutsos CMU SCS 15-415/615 85
