COSC 460 Lecture 19: Recovery 2

Professor Michael Hay
Fall 2018

Recap: Undo logging protocol

1. For each DB update, generate log record
2. Write ahead logging: before OUT(X), flush log records up to and including modifications of X.
3. Force: before <T_i, commit> to log, flush all pages dirtied by T_i

Recap: Redo logging protocol

First two same as undo logging.

1. For each DB update, generate log record
2. Write ahead logging: before OUT(X), flush log records up to and including modifications of X.
3. Before T_i commits, flush log
4. No steal: before OUT(X), must write <T_i, commit> to log

Recap: undo only and redo only logging

- Write head logging: ensures log is always “ahead” of DB.

- Disadvantage of undo only: if you can only undo, must **force** database to write *committed* changes.

- Disadvantage of redo only: if you can only redo, must *prevent* DB from writing *uncommitted* changes (“no steal”).

Undo/redo logging

- The *logging protocol* is simpler than that of undo or redo.

- The *recovery protocol* is more complex.

Undo/redo logging protocol

1. For each DB update, generate log record
2. Write ahead logging: before OUT(X), flush log records up to and including modifications of X.
3. Before T_i commits, flush log
4. ~~No steal: before OUT(X), must w~~ ~~<commit>~~ ~~to log~~

No force requirement (that undo requires)!

Drop “no steal” requirement (that redo requires)!

Undo/redo recovery protocol

- Two phases: redo, then undo
- Redo: “repeat history” including
  - … both winners *and losers*
  - … changes made during *any prior recoveries*
- Undo: rollback losers
  - … and log changes made during rollback

Initial version written on board. Full version is on handout.

Undo/Redo logging

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

Suppose a crash occurs and the log and DB are as shown. Use the undo/ redo recovery protocol to restore the DB.

Log

```
<T1 start>
<T2 start>
<T3 start>
<T3 B 8→12>
<T1 A 8→16>
<T2 A 16→32>
<T1 B 12→18>
<T3 commit>
```

DB

```
A: 32
B: 18
```

Why CLRs?

- CLR = “Compensation log record”
- Written during recovery by recovery manager
- Rationale: log records history, and undo is part of that history
- It simplifies recovery (simple example shown on board)

Undo/Redo logging

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

Revisit previous example, but suppose T2 started before T1.

Further suppose there is *another* *crash* during recovery before the second abort log record is written.

What goes wrong? Does this point to a flaw in our recovery protocol? Or is this example unrealistic?

Log

```
<T2 start>
<T1 start>
<T3 start>
<T3 B 8→12>
<T1 A 8→16>
<T2 A 16→32>
<T1 B 12→18>
<T3 commit>
```

DB

```
A: 32
B: 18
```

Checkpoints

- Rationale:
  - Simplify recovery: start from *last* checkpoint
  - Allow for log truncation
- Checkpoint itself must be executed carefully to ensure data isn't lost.

What happens during checkpoint?

- Lock buffer pool
- Flush log
- Flush all (dirty) pages from buffer pool *(why?)*
- Write ***checkpoint record*** to log and flush log
  - Includes list of *active* transactions
- Unlock buffer pool *(why important to lock/unlock?)*

Undo/redo recovery protocol

```
losers = { T_i such that T_i is active at checkpoint }
# REDO phase
for each log record from checkpoint to last
    if <T_i start>, add T_i to losers
    if <T_i commit>, remove T_i from losers
    if <T_i abort>, remove T_i from losers
    if <T_i X old->new>, then
        X = new
        W(X)
        OUT(X)
    if <CLR T_i X val>, then
        X = val
        W(X)
        OUT(X)
# UNDO phase
for each log record from last to first
    if <T_i X old->new> and T_i is a loser, then
        X = old
        W(X)
        OUT(X)
        write <CLR T_i X old>
    if <T_i start> and T_i is loser, then
        write <T_i abort>
        remove T_i from losers
        if losers is empty: break
```

Also provided on handout.

Undo/Redo logging

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

#	Log
1	`<T1 start>`
2	`<T1 A 1→5>`
3	`<T1 commit>`
4	`<T2 start>`
5	`<T3 start>`
6	`<T2 A 5→10>`
7	`<T3 B 8→16>`
8	`checkpt {T2,T3}`
9	`<T3 B 16→32>`
10	`<T3 commit>`
11	`<T2 B 32→12>`

Suppose a crash occurs and the log is as shown. Assume the undo/redo recovery protocol is being used.

A. What statements are redone?

B. What statements are undone?

C. Which lines of this log file can be safely truncated?

Recovery vs. rollback

- Recovery: response to system crash
- Rollback: response to transaction being aborted.
- Rollback behaves the same was as “undo” phase of recovery: aborted transaction = “loser”