2. Atomicity & Durability Using Shadow Paging

CSEP 545 Transaction Processing for E-Commerce

Philip A. Bernstein

Introduction

- To get started on the Java-C# project, you need to implement atomicity and durability in a centralized resource manager (i.e. a database).
- The recommended approach is shadowing.
- This section provides a quick introduction.
- A more thorough explanation of the overall topic of database recovery will be presented in a couple of weeks.

Review of Atomicity & Durability

- Atomicity - a transaction is all-or-nothing
- Durability – the results of a committed transaction will survive failures
- Problem
  - The only hardware operation that is atomic with respect to failure and whose result is durable is "write one disk block"
  - But the database doesn't fit on one disk block!

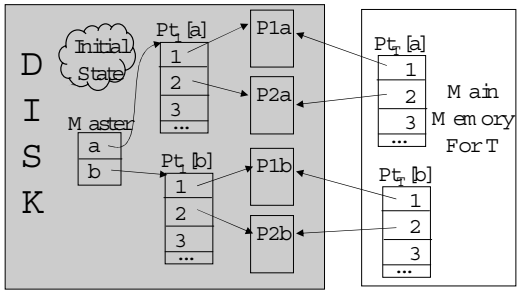
Shadowing in a Nutshell

- The database is a tree whose root is a single disk block
- There are two copies of the tree, the master and shadow
- The root points to the master copy
- Updates are applied to the shadow copy
- To install the updates, overwrite the root so it points to the shadow, thereby swapping the master and shadow
  - Before writing the root, none of the transaction's updates are part of the disk-resident database
  - After writing the root, all of the transaction's updates are part of the disk-resident database
  - Which means the transaction is atomic and durable

More Specifically...

- The database consists of a set of files
- Each file consists of a page table P and a set of pages that P points to.
- A master page points to each file's master page table.
- Assume transactions run serially. I.e., at most one transaction runs at any given time.
- Assume that for each page table the transaction has a private shadow copy in main-memory.

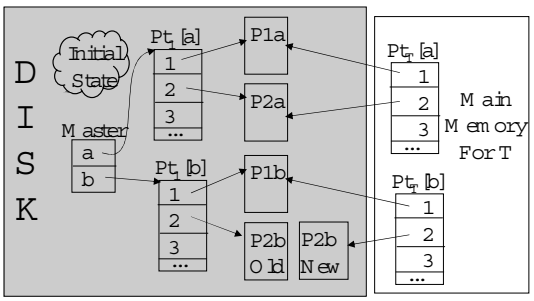
Initial State of Files a and b



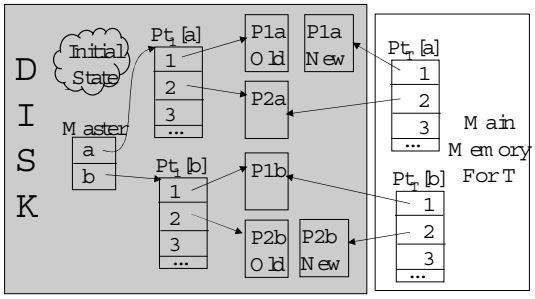
To Write a Page P_i

- Transaction writes a shadow copy of page P_i to disk (i.e. does not overwrite the master copy).
- Transaction updates its page table to point to the shadow copy of P_i
- Transaction marks P_i's entry in the page table (to remember which pages were updated)

After Writing Page P2b



After Writing Page P1a

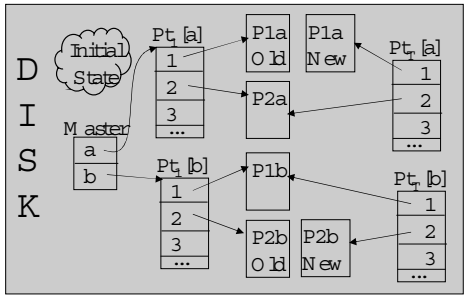


What if the System Fails?

- Main memory is lost
- The current transaction is effectively aborted
- But the database is still consistent

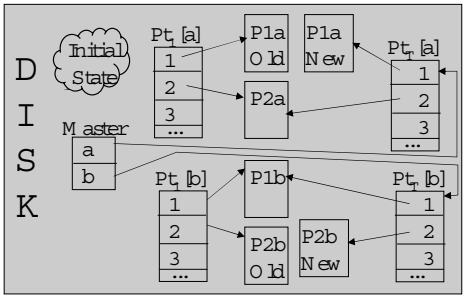
To Commit

1. First copy Pt_T [a] and Pt_T [b] to disk



To Commit (cont'd)

2. Then overwrite Master to point to the new Pt's.



Shadow Paging with Shared Files

- What if two transactions update different pages of a file?
  - If they share their main-memory shadow copy of the page table, then committing one will commit the other's updates too!
- One solution: File-grained locking (but poor concurrency)
- Better solution: use a private shadow-copy of each page table, per transaction. To commit T, do the following within a critical section:
  - For each file F modified by T
    - get a private copy C of the last committed value of F's page table
    - update C's entries for pages modified by T
    - store C on disk
  - Write a new master record, which swaps page tables for the files updated by T, thereby installing just T's updates

Managing Available Disk Space

- Treat the list of available pages like another file
- The master record points to the master list
- When a transaction allocates a page, update its shadow list
- When a transaction commits, write a shadow copy of the list to disk
- Committing the transaction swaps the master list and the shadow

Final Remarks

- A transaction doesn't need to write shadow pages to disk until it is ready to commit
  - Saves disk writes if a transaction writes a page multiple times or if it aborts
- Main benefit of shadow paging is that doesn't require much code
  - Was used in the Gemstone OO DBMS.
- But it is not good for TPC benchmarks
  - How many disk updates per transaction?
  - How to do record level locking?
- Most database products use logging.
  - Faster execution time, and more functional, but much more implementation.

Your Project

- You need not use the exact data structure presented here.
- In particular, you don't necessarily need a page abstraction.
- There are design tradeoffs for you to figure out.

References

- P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Chapter 6, Section 7 (pp. 201-204)
  - The book is downloadable from http://research.microsoft.com/pubs/ccontrol/
- Originally proposed by Raymond Lorie in "Physical Integrity in a Large Segmented Database" ACM Transactions on Database Systems, March 1977.