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).
- Recommend 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 - 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!

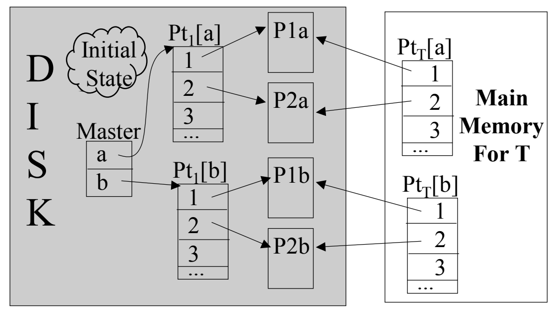
Shadowing in a Nutshell

- 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 a 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 no concurrency.
  I.e., one transaction runs at any given time.
- Assume the transaction has a private shadow copy of each page table.

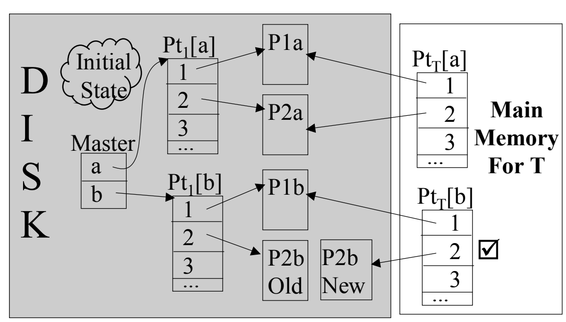
Initial State of Files a and b



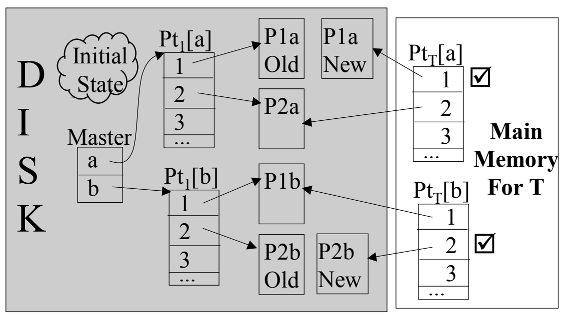
To Write a Page P_i

- Transaction writes a shadow copy of page P_i to disk
- 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

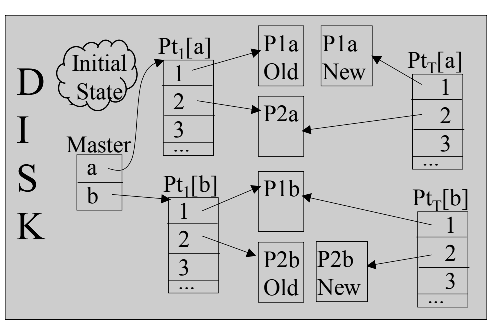


What if the System Fails?

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

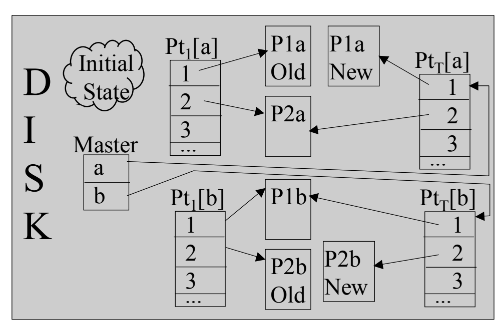
To Commit

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 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 copy of page table, per transaction. To commit T, *within a critical section*:
  - get a private copy of the last committed value of the page table of each file modified by T
  - update their entries for pages modified by T
  - store the updated page tables on disk
  - write a new master record, which installs just *T's* updates

4/3/03 13

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
- Commiting the transaction swaps the master list and the shadow

4/3/03 14

Final Remarks

- Don't need to write shadow pages to disk until the transaction is ready to commit
  - Saves disk writes if a transaction writes a page multiple times
- Main benefit is that doesn't require much code
- Used in the Gemstone OO DBMS.
- Not good for TPC benchmarks
  - count disk updates per transaction
  - how to do record level locking?

4/3/03 15

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.

4/3/03 16