Transactions

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Faculty evaluation forms
  - ***middle*** of class?
- Transactions
  - Text: 7.9, 17.3

Overview

- A different kind of critical section
- "ACID" Transaction Model
- Write-Ahead Logging
- Concurrency Control
- Distributed Transactions, 2-Phase Commit
- Camelot
- RVM

Very Critical Sections

- Critical section
  - Mutual exclusion
  - Progress
  - Bounded waiting
- Transaction
  - “Crash-proof” critical section
  - Moving money between bank accounts

A Simple Transaction

```
BEGIN_TRANSACTION
account1 = lookup(“293479238”);
account2 = lookup(“342342342”);
lock(account1); lock(account2);
if (account1->balance > 50)
  account1->balance -= 50;
  acount2->balance += 50;
  COMMIT;
else
  ABORT(“insufficient funds”);
END_TRANSACTION
```

“ACID” Transaction Model

- Intuition
  - Transaction succeeds (***crash-proof, forever***)
  - Or ***never happened***
- ACID = Atomic, Consistent, Isolated, Durable

Atomic

- Atomic = ***All or none***
- Failure of any step ***aborts*** transaction
  - Explicit – ABORT(char *reason)
  - Implicit – system crash
  - Distributed – ***any*** system crash
  - Aborted transactions ***have no visible effect***
- Committed transactions are ***completely visible***
- No inconsistent partial results

Consistent

- All transactions maintain ***database invariants***
  - Conservation of money
  - Every employee has a manager
- Split responsibility
  - Application transactions must be correct
  - Database may provide automatic checks

Isolated

- Concurrency is ***mandatory***
  - Cannot lock entire bank for each transaction
  - No global mutex
- Transactions must ***run*** concurrently
- Transactions must ***appear*** sequential
  - Serializability – *as if* some sequential order
  - Deposit happens ***before*** transfer or ***after*** transfer
    - Not lost between fetch and add/store

Durable

- Committed transactions are ***durable***
  - ***persistent, stable***
- Immune to crashes, disk failure, fire, flood
- As irrevocable as cash leaving the ATM

Storage

- Volatile – can “forget”
  - Cache, DRAM, /tmp
- Non-volatile – survives power outage
  - Disk, magnetic core memory, flash
- Stable – survives ***everything***
  - Store to a RAID array...
    - ...on each continent

Atomic/Durable conflict

- Atomic – don't store too soon
  - If error, must ***roll back*** to initial state
- Durable – must store ASAP
  - No step is durable before storage
- Resolution – ***write-ahead logging***

Write-Ahead Logging

- Log each intended mutation to disk
  - Transaction may "think" between modificatons
- sync()
- Apply modifications one by one
- sync()
- Ok to delete log
  - In theory, not in practice

Log Contents

```
BEGIN(tid=13)
WRITE(tid=13, rec=45, old=60, new=10)
BEGIN(tid=14)
WRITE(tid=14, rec=20, old=0, new=100)
COMMIT(tid=14)
ABORT(tid=13)
BEGIN(tid=15)
WRITE(tid=15, rec=1, old=0, new=1)
[system crash]
```

Key Operations

- undo(transaction)
  - Scan log...
  - Restore old values for transaction's writes
- redo(transaction)
  - Scan log...
  - Store new values for transaction's writes
- Crash recovery
  - redo() if BEGIN(t) and COMMIT(t), else undo()

Checkpoints

- Concept
  - Don't want to replay ***entire*** log during recovery
    - Most of it already written to database
- Approach – periodic ***checkpoint*** phases
  - Force log records from RAM to log disk
    - Not typically necessary before commit
  - Force mutations to database
  - Force CHECKPOINT to log disk

Checkpoints

- Result
  - Restart processing begins at newest checkpoint
- ***Warning*** about text
  - Checkpoint treatment incomplete
  - “Long-running” transactions may cross multiple checkpoints
  - Must be un-done even if no recent writes

Concurrency Control

- Recall isolation
  - Concurrent transactions sharing data must “make sense”
  - More formally, must ***appear sequential***
  - Serializability – *as if* some sequential order
- Consider balance transfer

```
read(account1);
write(account1); /* account1 -= x; */
read(account2);
write(account2); /* acount2 += x; */
```

Sensible Balance Transfer

T0	***T1***
`read(account1)`	
`write(account1)`	
`read(account2)`	
`write(account2)`	
	`read(account1)`
	`write(account1)`
	`read(account2)`
	`write(account2)`

Non-sensible Balance Transfer

T0	*T1*
`read(account1)`	
	`read(account1)`
`write(account1)`	
	`write(account1)`
`read(account2)`	
`write(account2)`	
	`read(account2)`
	`write(account2)`

Single debit, double credit – ***money is created!***

Conflicting Operations

- Operations ***conflict*** if
  - Access same data item
  - One or more write operations
- Serializability rule
  - Ok to interleave transaction operations when...
  - Start with a serial schedule
  - Swap ***non-conflicting*** operations

Serializable Balance Transfer

T0	*T1*
`read(account1)`	
`write(account1)`	
	`read(account1)`
	`write(account1)`
`read(account2)`	
`write(account2)`	
	`read(account2)`
	`write(account2)`

Serialization Approaches

- Locking protocol
  - Shared and exclusive locks (reader/writer)
  - Growing phase, then shrinking phase, then commit
- Timestamp protocol
  - New transactions assigned timestamps
  - Data read-stamped, write-stamped by transactions
  - read(tstamp = 45, data-write-stamp = 55)
    - Necessary value was overwritten by another transaction
    - Must abort or restart

Distributed Transactions

- Concept
  - Balance transfer between branches
    - ...on different continents
  - What if one branch crashes?
- Approach
  - Local ***transaction manager*** per branch
    - Traditional logging, recovery
  - Single ***transaction coordinator***
    - Manages distributed commit processing

Two-Phase Commit - 1

- Transaction completes all operations
- Coordinator forces `PREPARE(tid)` to its log
  - Announces `PREPARE(tid)` to all sites
- Each site make go/no-go decision
  - Forces `READY(tid)` or `NO(tid)` to log
  - Forces transaction operations to log
  - Answers coordinator

Two-Phase Commit - 2

- Coordinator gathers responses
  - Any `NO` means failure
  - Timeout means failure
- Coordinator forces decision to its log
  - `COMMIT(tid)` or `ABORT(tid)`
- Coordinator transmits verdict to all sites
- Each site logs, obeys

Site Restart

- If `COMMIT(tid)` or `ABORT(tid)`, obvious
- If no `READY(tid)`, abort
- If `READY(tid)` in log
  - Any site has `COMMIT` or `ABORT`: obvious
  - Any site has no `READY`
    - Coordinator failure? Abort
  - Everybody `READY`? ***Need*** coordinator

Camelot Project

- CMU CS project, late 80's
- Distributed transaction system
- ***Transactional virtual memory***
  - No external “database records”
  - All data in persistent ***transactional*** memory
- Made heavy use of Mach
  - Threads
  - “External pager” handled page faults, flushes

Camelot Evaluation

- Exciting, versatile, usable system
  - Mere mortals wrote distributed transactional applications
  - Did not become a product
- "Research system" issues
  - Performance, Mach dependence
- Is transactional memory the right model?
  - Database > 4 gigabytes?
  - Upgrade to a new processor architecture?

RVM Library

- Developed by CMU CS Coda project
- Goal – “Camelot light”
  - Camelot task modularity was slow
  - Camelot required Mach
- Design
  - Toss distributed and nested transactions
  - Application manages concurrency control
  - OS manages media failure

RVM Library

- RVM provides
  - Atomicity (logging, restart)
  - Fine-grained control over log behavior
    - Some transactions may not need ***immediate*** log force
- Portable
  - NetBSD, FreeBSD, Linux
  - Windows
- ftp://ftp.coda.cs.cmu.edu/pub/rvm/

Summary

- Transaction – Sequence of operations
  - Atomic, consistent, isolated, durable
- Transaction – building block
  - Unifying concept for system building
- Write-ahead logging
  - Log-replay during system restart
  - Checkpoints
- Distributed transactions - 2PC

Further Reading

- Transaction Processing: Concepts and Techniques
  - Jim Gray @ Digital Equipment

    [insert moment of silence for DEC]
  - Andreas Reuter @ University of Stuttgart
  - 1993
  - Definitive
- Lightweight Recoverable Virtual Memory
  - Satyanarayanan, Mashburn, Kumar, Steere, Kistler
  - SOSP 14 (1993)