Transaction Processing

- normal systems lose their state when they crash
- many applications need better behavior
- today's topic: how to get "industrial strength" fault protection
  - try to prevent failures
  - deal intelligently with failures

Example: Banking

- consider a bank's computer
- to transfer money from savings to checking
  - subtract from savings
  - add to checking
- what if machine crashes in the middle?

Transactions

- transaction: a sequence of operations that is
  - atomic: either executes completely, or not at all
  - consistent: leaves the system in a sensible state
  - isolated: intermediate states invisible to others
  - durable: once completed, its effects can't be undone by a failure
- "ACID" properties
- locking gives you C and I, but not A and D

Transaction-Processing

- examples
  - airline reservation records
  - business inventory and billing records
  - personal finance software (like Quicken)
  - most database programs
- before PCs, transaction processing consumed more cycles than any other kind of computation

Example, with Transactions

```
void transferMoney(Customer cust, Money amount) {
        Transaction t;

        do{
                t = beginTransaction();
                Money sav = transRead(savings[cust], t);
                Money check = transRead(checking[cust], t);
                sav -= amount;
                check += amount;
                transWrite(savings[cust], sav, t);
                transWrite(checking[cust], check, t);
        }while(endTransaction(t) == Abort);
}
```

Transaction Operations

- "official" version of all values lives on disk
  - values cached in memory
- beginTransaction: starts transaction and gets a transactionID
- endTransaction: either Commits or Aborts
  - commit makes all writes durable
  - abort discards all writes
- transRead, transWrite
  - write don't take effect until Commit

Aborting Transactions

- Why do transactions abort?
  - client explicitly asks for abort
  - server crashed during transaction
  - transaction required a resource that wasn't available
  - transaction conflicted with another transaction
    - » both wrote the same location, or one wrote it and one read it
    - » can abort either transaction, as long as progress is guaranteed

Implementing Transactions

- start with the non-distributed case
- each transaction keeps a log of its writes
  - record holds address written, and value
- transWrite operation creates log record, but doesn't write to the real data location

Implementation: Non-Distributed

- on abort, discard the transaction's log
- on commit, make all of the logged writes occur as an indivisible action
  - locking prevents other transactions from seeing partially-written data
  - but what about a crash?
    - » might have only some of the data written to disk before the crash

Intentions List

- disk lets you write one block at a time
- to make several writes atomic
  - write an "intentions list" to a well-known place
    - » says what writes you are going to do
  - do the intended writes
  - erase the intentions list
- after reboot, look for intentions list
  - if you find one, do the writes on it, then erase it

Committing with Intentions List

- to commit a transaction
  - copy log to intentions list
  - mark transaction as committed
  - do writes and erase intentions list
- optimization: restart app once intentions list is written, do the rest in background
- further optimization: let many intentions lists build up before writing them all out at once

The Stable Log

- an append-only file in stable storage
  - contains intentions lists for many transactions
  - "self-explanatory" format
  - lives at well-known location
- "replay" stable log after crash recovery
  - perform all intended writes
- can "trim" a transaction from the front of the stable log after all of its writes have been done

Distributed Transactions

- a transaction may invoke operations on several servers
- clients and servers may crash and recover independently
- extensions to non-distributed mechanism:
  - transactional RPC
  - distributed logging
  - distributed commit/abort decision

Transactions and RPC

- transactionIDs must be globally unique
  - embed originating process address in them
- every RPC involved in a transaction must pass transactionID to the server
- server does transRead and transWrite rather than ordinary read and write

Distributed Logging

- each machine keeps its own log for the transaction
- each machine keeps its own stable log
- machine that initiated transaction acts as "coordinator" for that transaction
  - all participants know who manager is
  - manager has a list of all participants

Distributed Commit/Abort

- this is the hard part!
  - must agree whether to commit or abort
    - » abort if anybody wants to abort
    - » commit if everybody wants to commit
  - must all commit "simultaneously"
  - protocol must work even if participants crash and recover during commit/abort process
- usual approach is "two-phase commit"

Two-Phase Commit

- phase 1: manager asks participants, "Do you want to commit?"
  - participants answer
- phase 2: manager informs all participants of decision
  - participants acknowledge and perform the appropriate actions
- abort if anybody fails before responding in phase 1

Two-Phase Commit and Failures

- anybody who votes to abort can forget about the transaction; it will certainly abort
- anybody who votes to commit must be prepared to commit
  - must store intentions list in stable storage, to protect against crash
  - but label intentions list as "pending" since transaction might still abort
    - » relabel as "committed" if transaction commits

Recovery in Two-Phase Commit

- cases to handle:
  - non-manager crashes between phases:
    - » recovering machine finds a pending transaction in its stable log
    - » asks manager what happened to the transaction
  - manager crashes between phases
    - » all participants wait for manager to come back
    - » manager re-does phase 1 voting

Advanced Topics

- concurrency control
  - what if two transactions want to use the same data item?
- nested transactions
  - what if a transaction wants to do an atomic sub-transaction?