Recovery

• First order of business:
  – Failure Model
Types of Failures

Events — Desired
Undesired — Expected
Unexpected

Our failure model

- CPU
- Processor
- Memory
- Disk

© 2021 Christopher W. Clifton
Desired events: see product manuals….

Undesired expected events:
System crash
  - memory lost
  - cpu halts, resets
    that’s it!!

Undesired Unexpected: Everything else!

---

Failure Classification

- **Transaction failure**: transaction cannot complete due to some internal error condition
  - **Logical errors**: transaction cannot complete due to some internal error condition
  - **System errors**: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

- **System crash**: a power failure or other hardware or software failure causes the system to crash.
  - **Fail-stop assumption**: non-volatile storage contents are assumed to not be corrupted by system crash
    - Database systems have numerous integrity checks to prevent corruption of disk data

- **Disk failure**: a head crash or similar disk failure destroys all or part of disk storage
  - Destruction is assumed to be detectable: disk drives use checksums to detect failures
Undesired Unexpected: Everything else!

Examples:
- Disk data is lost
- Memory lost without CPU halt
- Sun goes supernova

Is this model reasonable?

**Approach:** Add low level checks + redundancy to increase probability model holds

E.g., Replicate disk storage (stable store)
- Memory parity
- CPU checks
Storage Structure

- **Volatile storage:**
  - Does not survive system crashes
  - Examples: main memory, cache memory

- **Nonvolatile storage:**
  - Survives system crashes
  - Examples: disk, tape, flash memory, non-volatile RAM
  - But may still fail, losing data

- **Stable storage:**
  - A mythical form of storage that survives all failures
  - Approximated by maintaining multiple copies on distinct nonvolatile media
  - See book for more details on how to implement stable storage

Stable-Storage Implementation

- Maintain multiple copies of each block on separate disks
  - copies can be at remote sites to protect against disasters such as fire or flooding.

- Failure during data transfer can still result in inconsistent copies: Block transfer can result in
  - Successful completion
  - Partial failure: destination block has incorrect information
  - Total failure: destination block was never updated

- Protecting storage media from failure during data transfer (one solution):
  - Execute output operation as follows (assuming two copies of each block):
    1. Write the information onto the first physical block.
    2. When the first write successfully completes, write the same information onto the second physical block.
    3. The output is completed only after the second write successfully completes.
Protecting storage media from failure (Cont.)

- Copies of a block may differ due to failure during output operation.
- To recover from failure:
  1. First find inconsistent blocks:
     1. **Expensive solution**: Compare the two copies of every disk block.
     2. **Better solution**:
        - Record in-progress disk writes on non-volatile storage (Flash, Non-volatile RAM or special area of disk).
        - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these.
        - Used in hardware RAID systems
  2. If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy. If both have no error, but are different, overwrite the second block by the first block.
Recovery Algorithms

- Suppose transaction $T_i$ transfers $50$ from account $A$ to account $B$
  - Two updates: subtract 50 from $A$ and add 50 to $B$
- Transaction $T_i$ requires updates to $A$ and $B$ to be output to the database.
  - A failure may occur after one of these modifications have been made but before both of them are made.
  - Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
  - Not modifying the database may result in lost updates if failure occurs just after transaction commits
- Recovery algorithms have two parts
  1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
  2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

Recovery and Atomicity

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.
- We study log-based recovery mechanisms in detail
  - We first present key concepts
  - And then present the actual recovery algorithm
- Less used alternative: shadow-copy and shadow-paging (brief details in book)