Lecture 16:

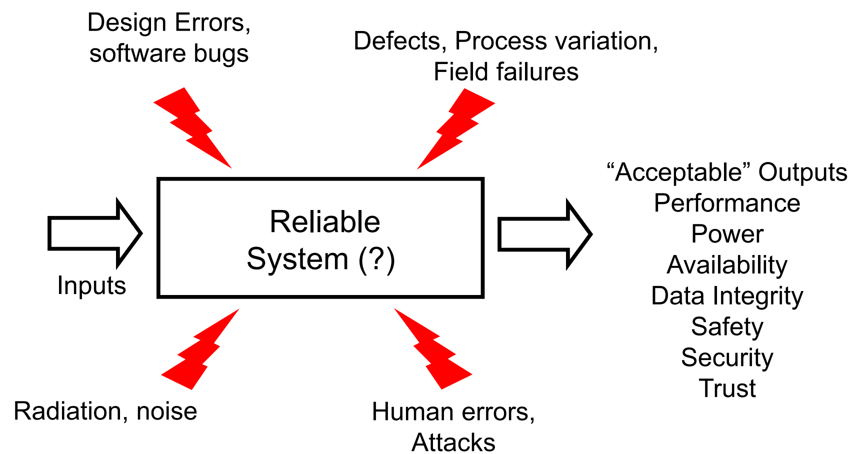
Reliable Systems

Department of Electrical Engineering
Stanford University

http://eeclass.stanford.edu/ee282

Announcements

Are Systems Reliable?



Who Needs Reliable Systems?

- Classical view
  - High end enterprise systems, airplanes, space missions, …
- Reality: everybody needs reliable systems
    - Enterprise, desktop, mobile, …
  - Computers control important aspects of our lives
  - Certain computers operate in harsh environments
  - Hardware & software are becoming too complex to test
  - Several types of errors are becoming too common
  - Design reuse for both volume & high end
- But reliability comes at a cost…
  - Research challenge: enterprise reliability @ desktop cost

Definitions

- **Fault**: an undesirable event
  - E.g. software bug, design error, alpha particle, …
  - A fault may create an error
- **Error**: the negative result of a fault
  - E.g. a variable gets an incorrect value
  - An error may propagate when exercised
- **Failure**: when service is unavailable or data integrity is lost
  - The user can really tell

- An error is manifestation in the system of a fault
- A failure is manifestation on the service of an error
- At the end, it's all about avoiding failures…

Types of Faults

- Based on duration
  - Permanent
    - Defects, bugs, out-of-range parameters, wear-out, …
  - Transient (temporary)
    - Radiation issues, power supply noise, EMI, …
  - Intermittent (temporary)
    - Oscillate between faulty & non-faulty operation
    - Operation margins, weak parts, activity, …

- Based on origin
  - Software, hardware
  - Design, operation, environment
  - …

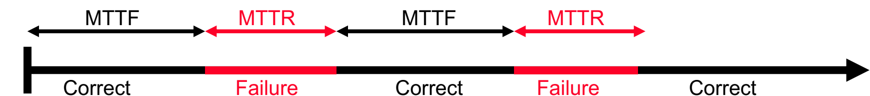
Metrics

- **Reliability**: probability of correct output up to time t
  - Common poison model: $R(t) = e^{-\lambda t}$, where is λ failure rate
  - Common assumption for component failure rates
    - Independent & exponentially distributed
    - FR = SUM(λ_i) for all modules i in the system

- **MTTF**: mean time to failure
  - Time to produce first incorrect output
  - With poison model, MTTF is $1/\lambda$

- **MTTR**: mean time to repair
  - Time to detect and repair a failure

Availability



- **Availability**: probability of correct output at time t
  - Steady state availability = MTTF / (MTTF + MTTR)

- For higher availability, you need one of
  - Very high MTTF (fault-tolerant computing)
  - Very low MTTR (recovery-oriented computing)

- Availability often quoted in 9s
  - E.g. the telephone system has five 9s availability
  - 99.999% availability or 5 minutes of downtime per year

Metrics Continued

- Other metrics
  - MTBF: mean time before failure (MTTF+MTTR)
  - FIT: failure in time (10^9/MTBF)
  - AFR: annual failure rate (FIT * 8760/10^9)
- MTTF can be misunderstood
  - What does MTTF = 20 years mean?
    1. The system does not fail in 20 years?
    2. If you have 20 systems, you'll get one failing per year?

Not All Faults Lead to Errors or Failures

- Fortunately, most errors are benign
- Memories
  - Errors in unused memory bits
  - Errors in memory bits soon to be overwritten
- Logic and processing
  - Errors in idle units
  - Errors in NOP instructions
  - Errors in performance enhancing instructions (e.g. prefetch)
  - Errors in mispredicted instructions
- Application
  - Errors in unused or unimportant variables

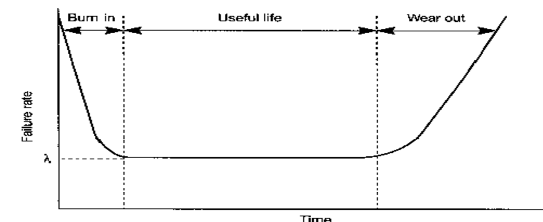
Approaches to Reliable Systems

1. Do nothing ☹
   - Rely on components not failing too often for your application
   - Costs nothing but you get what you paid for
2. Fault avoidance
   - Design system to be as robust as possible (avoid failures)
   - Conservative design, thorough validation & test, design to avoid errors, ..
3. Fault tolerance
   - Detect and isolate errors during operation
   - On-line recovery and self-repair from failures

- Both avoidance and tolerance come at a cost!

Fault Avoidance

- Build systems to avoid failures to begin with
  - Conservative design (large margins for timing, power, …)
  - Comprehensive testing & debugging
  - Verification (sometimes formal)
  - Preventive maintenance (may include downtime)
- Infant mortality screen for chips (aka burn-in)



Fault Tolerance

- Approaches
  - Mask errors
  - Detect and recover from errors
- Fault tolerance is based on redundancy
  - Replicated logic, redundant storage, error coding in memory, …
  - No single-point of failure…
- Why redundancy works
  - If a component is unavailable at time t with probability P(t)
  - Then N-way redundancy reduces the probability to $P(t)^N$
  - If P(t)=0.1 and N=3, then new P(t) is 0.001
- The key trade-off: redundancy vs cost
  - Look at typical error patterns & reliability requirements

General Steps for Handling Errors

1. Error detection & confinement
2. Error masking or retry
   - Most transient errors will be removed by this point
3. Diagnosis
   - Which is the defective part?
4. Reconfiguration
   - Assuming hot spares are available to handle permanent errors
5. State recovery & restart
6. Repair or replacement
   - If no more hot spares are available
7. System reintegration

Example of Reliability Techniques

- Processing & storage systems
  - Error correcting codes, N_of_M and standby redundancy, TMR & voting, watchdog timers, reliable storage (RAID, mirrored disks), checkpointing and rollback
- Networking
  - CRC on messages , acknowledgment, watchdogs, heartbeats, consistency protocols
- System software
  - Memory management, detection of process failures, hooks to support software fault tolerance for application
- Application level software
  - Checkpointing and rollback, application replication, software, voting (fault masking), process pairs, robust data structures, recovery blocks, N-version programming,

Rest of Lecture: Practical Techniques for Improving Fault-tolerance

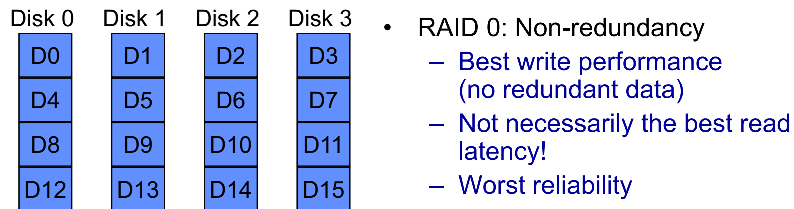
- We'll go over example techniques for
  - Storage systems
  - Memory systems
  - Processing systems
  - Interconnects
- Goal: understand applicability & cost
  - Which faults do they catch or miss?
  - How do they compare to alternatives?
- Note: these techniques generalize
  - With many system types, at many scales, …
  - Apply them creatively

Reliability in Storage Systems

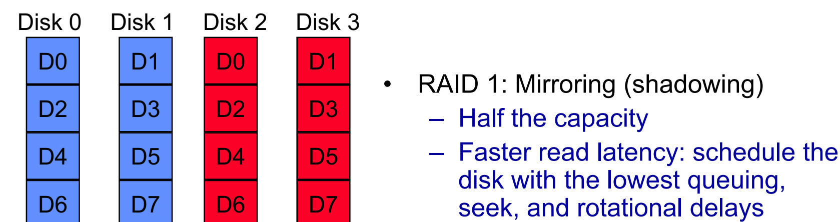
RAID: Dealing with Faults in Hard Disk

- HDs use error correcting codes internally to masks certain errors
  - However, you can still have a whole disk fail
    - Mechanical head crashes, etc
  - Especially important in large storage systems with many disks
    - If MTTF = 200,000 hours (23 years) for a single disk, MTTF = 2000 hours (3 months) for an array of 100 disks
- Solution: redundant arrays of inexpensive disks (RAID)
  - A collection of disks that behaves like a single disk with
    - High capacity, high bandwidth, high reliability disk
  - Key idea in RAID: error correcting information across disks
    - Similar to chipkill in DRAM
  - Many organizations; two distinguishing features:
    - The granularity of the interleaving (bit, byte, block)
    - The distribution of redundant information
  - Patterson classification: RAID levels 0 to 6

RAID Levels 0 and 1: Block Duplication

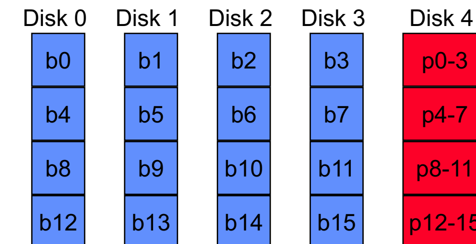


- RAID 0: Non-redundancy
  - Best write performance (no redundant data)
  - Not necessarily the best read latency!
  - Worst reliability



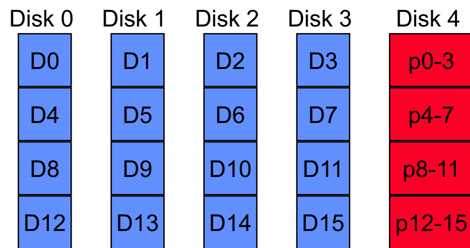
- RAID 1: Mirroring (shadowing)
  - Half the capacity
  - Faster read latency: schedule the disk with the lowest queuing, seek, and rotational delays

RAID Level 3 Bit Interleaving and Simple parity



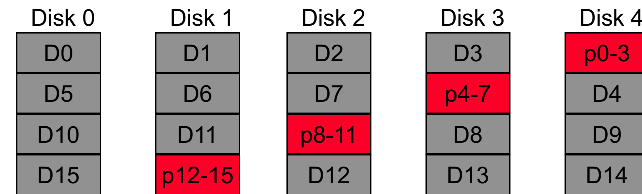
- Use 1 additional disk to keep track of parity information
- Unlike memories, disk controllers can detect which disk has failed
  - Encoding in disk has redundancy of its own
  - So a single parity bit is enough to correct any single-bit failure
- High bandwidth: all disks deliver data in parallel
- Good for large block transfers (images, etc.)

RAID Level 4
Block Interleaving and Simple Parity



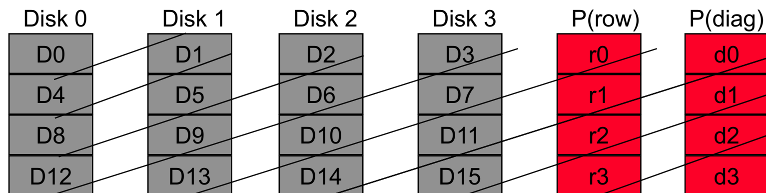
- Like RAID 3, but block interleaving
- Can recover from any single disk or block failure
- Read requests smaller than a block only access a single disk
  - Multiple requests can be executed in parallel
- Write requests smaller than a block can update parity using only the old data and old parity blocks

RAID Level 5
Block Interleaving and Distributed Parity



- Like Level 4, but eliminates dedicated parity disk
  - Not a bottleneck for multiple simultaneous writes
  - Allows 5 (not 4) simultaneous read operations
- Best small read, large read, and large write performance of any RAID
  - The “left symmetric” pattern shown has the best performance:
    - For sequential accesses, access each disk once before accesssing any disk twice
- Small write performance worse than, say, mirroring
- Most common RAID implementation

RAID Level 6
Multiple Types of Parity



- Protect against 2 or more disk failures
- Textbook example: row-diagonal parity
  - Example separates parity disks for clarity; can be distributed like RAID 5
  - Another scheme: XOR + another function for row parity
- Implementation can be slow, requiring dedicated hardware

- Good for all reads, bad for small writes

Reliability in Memory Systems

Permanent Faults

- Stuck-at-0 or stuck-at-1 memory bits
  - Often as we increase memory capacity or decrease bit area
- Solution: redundant memory resources
  - Rows/columns in arrays, whole arrays, …
  - Use build-in self-test (BIST) engines to detect stack bits
    - They generate proper access patterns for testing purposes
  - Configure row/column decoders to map out thoe with defective bits
    - By blowing fuses that implement metal connections for the decoders
  - Done during manufacturing test
- Applicable to both DRAM and SRAM memories

Memories: Transient Faults

- Bits flip 0→1 or 1→0
  - Soft errors, noise, …
- Solution: extra bits per word & error detecting/correcting codes
- Parity: a simple error detecting code
  - Add a 9th bit to each 8-bit word,
  - Calculate parity by XOR-ing the 8 bits in word
    - E.g. if word is 01001101, the parity is 0
  - Allows to detect odd number of bit errors
    - But cannot correct any mistake
  - Error detected when word is read by recalculating parity
    - And comparing to the one stored in the 9th bit

Error Correcting (ECC) and Error Detecting (EDC) Codes

- Hamming codes:
  - For every N bits of data, use P additional parity bits
    - Each parity bit is calculated on a subset of the N bits
  - Use parity bits to detect x errors and correct y errors (y<x)
- Common case: SECDED (y=2, x=1)
  - Can buy DRAM chips with (64+8)-bit words to use with SECDED
- Example: for N=16 can build a SECDED code (y=2, x=1) with P=5
    - P0 = D15 ^ D13 ^ D11 ^ D10 ^ D8 ^ D6 ^ D4 ^ D3 ^ D1 ^ D0
    - P1 = D13 ^ D12 ^ D10 ^ D9 ^ D6 ^ D5 ^ D3 ^ D2 ^ D0
    - P2 = D15 ^ D14 ^ D10 ^ D9 D8 ^ D7 ^ D3 ^ D2 ^ D1
    - P3 = D10 ^ D9 ^ D8 ^ D7 ^ D6 ^ D5 ^ D4
    - P4 = D15 ^ D14 ^ D13 ^ D12 ^ D11

The Idea behind Hamming Codes

- Hamming distance: the # of bits that are different between two words
  - 0001 and 1000: hamming distance D=2
  - We can detect any single-bit error given these two words
    - No single bit can switch 0001 to 1000
- In general, we use the P additional bits to construct codes were all valid words are at least D bits apart
  - We can detect D-1 errors
  - To correct x errors, we need the D >= 2x+1
    - If x=1, we must have $2^P > N+P+1$
- There are many different types of Hamming codes
  - N, P, …
  - Type of errors handled (bursts, isolated etc)

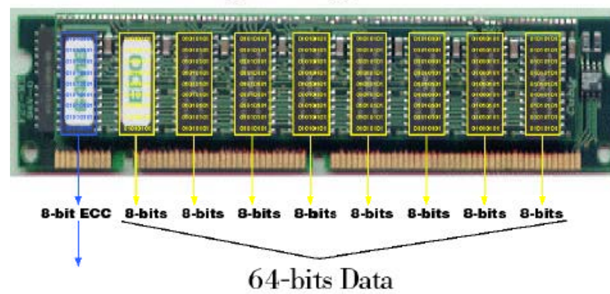
ECC/EDC Issues

- ECC DRAMs
  - Provide additional storage (74-bit words)
  - But logic for ECC check/generation is in memory controller
- Performance issues
  - Every subword write is a read-modify-write
  - Necessary to update the ECC bits
- Reliability issues: double-bit errors
    - Cosmic rays can affect multiple neighboring bits
  - Likely if we take a long time to re-read some data
  - Solution: use scrubbing
    - A background engine periodically reads memory words, detects and corrects single-bit error before 2nd error becomes likely

ECC DRAM DIMM

- Conventional approach: a DIMM with ECC DRAM chips
  - Data + error coding in same DRAM chip
  - Can detect/correct errors within a chip
- Problems:
  - What if a chip gets a double-bit error?
  - What if a whole chip fails?
    - Interconnect, interface, or clocking logic error
- Solution: chipkill DIMMs
  - Instead 8 use 9 regular DRAM chips (64-bit words)
  - Don't do ECC within chips, do it across chips

ChipKill DRAM DIMM



- Can tolerate errors both within and across chips
- You can also do RAID across DIMMs…

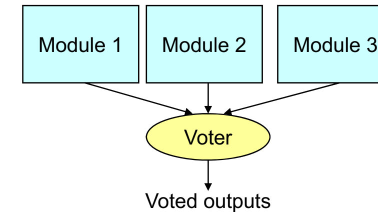
Reliability in Interconnects

- Use error correcting codes
  - Similar to what we did for DRAM chips
  - Receiver can recreate bits that have been inverted
- Use error detecting codes and retransmissions
  - Receiver detects error and request retransmission
    - Requires buffering at the sender side
  - An ack/nack protocol is typically used
    - To indicate when the receiver received correct data (or not)
  - Time-out mechanisms to deal with the case of lost messages
    - Error in control signals or with acknowledgements
- Permanent faults?
  - Use network with path diversity

Reliability in Computing

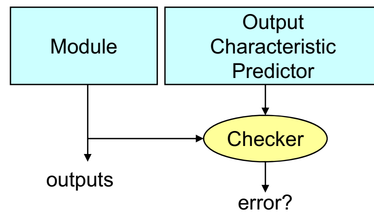
- For systems ranging from ALUs to full computers
- Not as simple (or cost effective) as memory
- Techniques
  - Triple module redundancy (TMR)
    - Or more than triple…
  - Concurrent error detection (CED)
    - In various hardware & software forms
  - Checkpoint & restore
    - Also called Backward Error Recovery (BER)
- Consider two issues: detection & recovery
  - Not always clearly separated…

Triple Module Redundancy (TMR)



- Advantages
  - Can fully mask 1 failed component
    - Any type of fault
  - Generalizes to N-MR (can tolerate [N/2-1] faulty components)
- Disadvantages
  - High implementation cost
  - Reliability of voter?
  - Synchronization of modules?

Concurrent error detection (CED)



- CED idea
  - Used a 2nd module to detect errors
    - Any type of fault again
  - Once error is detected, use re-execution for correction
    - Works with many transient errors
    - Repeated error signal a more systematic problem

CED Details

- What can the 2nd module be
  - A replica of the computation module
    - Checker is an equality comparator
  - A parity predictor
    - Checker compares actual to predicted parity
  - Many others
- Common mode errors
  - An error affects both modules the same way
  - Solutions: design diversity
    - Use different implementations of same function
    - N version programming
    - Use different processors (in Boeing and Airbus)

CED Issues

- At what level do you apply CED?
  - Compare ALU outputs
  - Compare Register updates
  - Compare L1/L2/L3 traffic?
- Bandwidth requirement vs. detection latency tradeoff
  - If you compare all ALU outputs: fast detects, need lots of BW
  - If you compare all L2 cache traffic: low BW, delayed detection
- A solution
  - Hash all chip updates using a CRC code (e.g. 16-bit)
  - Compare CRCs across two modules on every cycle
    - Low probability of masking an error
    - Low bandwidth usage
    - Fast fault detection (but coarse-grain)

Checkpoint & Restore (Backward Error Correction)

- Basic idea
  - Periodically get a system state checkpoint (registers, memory, …)
    - Checkpoint must be stored in “protected storage”
  - During execution, try to detect errors
  - On error, roll back to checkpointed state and restart
    - Works for transient faults
    - For permanent errors, first reconfigure and then restore
- Advantages
  - A general concept applicable to many systems and types of errors
- Disadvantage
  - Recovery carries a performance penalty
  - Not ideal when there is long error detection latency
    - If detection time > checkpoint interval
  - Some part of the system state may be unrecoverable
    - E.g., if an error affects an external state not checkpointed

Creating Checkpoints of System State

- Naïve approach: make a complete copy of system state
  - Takes too much time and too much space
- Incremental checkpoints
  - Keep track (log) of state changes since previous checkpoint
    - Checkpoint entries: address, old value, new value
  - On fault, undo changes to establish checkpoint
  - Reduces space & time requirements to restore

- Checkpoint challenges (particularly in parallel systems)
  - Checkpoint frequency vs detection latency
  - Number of active checkpoints
  - Synchronization (may need to stop the world)
  - Dirty data in caches

Reliability for the Full System

- IO, cooling, power supply…
  - N+1 redundancy (no single point of failure)
  - Service processor to monitor control system health
- Power subsystem example