Fault Tolerance

Content taken from the following:
“Distributed Systems: Principles and Paradigms” by Andrew S. Tanenbaum and Maarten Van Steen (Chapter 8)
Various online sources, including github.com/donnemartin/system-design-primer
Fault tolerance

- Desirable system properties
- Failure models
- Dealing with failure
Desirable system properties

• We want dependable systems
  – Available: ready for use at any given time
  – Reliable: runs continuously without failure
  – Safe: nothing catastrophic happens upon failure
  – Maintainable: easy to repair
  – Similar to definitions for dependable software (CS 345)
Problem

• Inherent tension between:
  - **Consistency**: reads see previous writes ("safety")
  - **Availability**: operations finish *relatively quickly* ("liveness")
  - **Partition tolerance**: failures don't affect correctness

*Systems design involves tradeoffs*
Problem

Which of the following is most important in a distributed system?

- A. Consistency
- B. Availability
- C. Partition tolerance
CAP Theorem

A system cannot be simultaneously consistent (C), available (A), and partition-tolerant (P)
- We can only have two of three
- In a non-distributed system, P isn't needed
  - Tradeoff: latency vs. consistency ("PACELC Theorem")
- In a distributed system, P isn't optional
  - Thus, we must choose: CP or AP
  - I.e., consistency or availability

Original conjecture by Eric Brewer: http://dl.acm.org/citation.cfm?id=822436
Formal theorem: http://dl.acm.org/citation.cfm?id=564601
Problem

- Which of the following is **least** important in a distributed system?
  - A. Consistency
  - B. Availability
  - C. Partition tolerance
Consistency

• Usual choice: compromise on consistency
  – **Strong consistency**: reads see all previous writes (sequential consistency)
    • Alternatively, continuous w/ short interval
    • Causal consistency: reads see all causally-related writes
  – **Eventual consistency**: reads eventually see all previous writes (continuous w/ long interval)
    • E.g., "guaranteed convergence"
  – **Weak consistency**: reads may not see previous writes
    • E.g., "best effort"
Availability

- **Active-passive / master-slave (asymmetric)**
  - Master server handles all requests
  - Backup/failover server takes over if master fails

- **Active-active / master-master (symmetric)**
  - Multiple master servers share request load
  - Load re-balances if one fails
Availability

- The JMU CS software mirror consists of two servers mirror1 and mirror2. At any given point, one is designated “primary” and handles all incoming traffic. If it fails, the other server will take over as primary. Which availability model is this closest to?
  - A. Active-passive
  - B. Active-active
  - C. Passive-passive
Failure models

- Sometimes, consistency/availability tradeoff decisions depend on the failure model:
  - What kinds of failures happen?
  - How often do they happen?
  - What are the effects of a failure?
  - At what level of abstraction does the failure take place?
  - How hard is it to debug a failure?
Kinds of failures

- **Soft vs hard failures**
  - **Soft failure**: a.k.a. silent data corruption (SDC)
    - Often corrected by hardware
  - **Hard failure**: a component of a system stops working

- **Hard failures in a non-distributed system are usually fatal**
  - The entire system must be restarted

- **Hard failures in a distributed system can be non-fatal**
  - **Partial failure**: a failure of a subset of the components of a distributed system
  - If the system is well-designed, it may be able to recover and continue after a partial failure
Kinds of failures

• A buffer overflow bug causes inadvertent data corruption. What is this an example of? (select all that apply)
  - A. Soft failure
  - B. Hard failure
  - C. Partial failure
Kinds of failures

• One of the JMU cluster nodes goes offline due to a faulty power supply. What is this an example of? (select all that apply)
  - A. Soft failure
  - B. Hard failure
  - C. Partial failure
Measuring failure

- **Failure rate** ($\lambda$): failures per unit of time
- **Mean Time Between Failures (MTBF)** $= 1 / \lambda$
  - Assumes constant failure rate
- **Failures In Time (FIT)** $= $ failures expected in one billion device-hours
  - $MTBF = 1e9 \times 1/FIT$

On a 10 million core machine, 1 FIT means once every 100 hours or **once every ~4.2 days**!
Measuring failure

If a JMU cluster hard drive dies on average every 5 years, what is the failure rate?

- A. 0.05 failures/yr
- B. 0.2 failures/yr
- C. 0.5 failures/yr
- D. 2.0 failures/yr
- E. 5.0 failures/yr
Effects of failure

- **Crash**: the system halts
- **Omission**: the system fails to respond to requests
- **Timing**: the system responds too slowly
- **Response**: the system responds incorrectly
- **Arbitrary** failure: anything else (unpredictable!)
  - Sometimes called "Byzantine" failures if they can manifest in such a way that prevents future consensus
Levels of failure

• Some systems distinguish between failure levels:
  – A failure occurs when a system cannot meet its specification
  – An error is the part of a system's state that leads to a failure
  – A fault is the low-level cause of an error
  – Most common source of faults: memory or disk storage

• If a system can provide dependable services even in the presence of faults, that system is fault-tolerant
Debugging faults

- **Permanent** faults reproduce deterministically
  - These are usually the easiest to fix
- **Intermittent** faults recur but do not always reproduce deterministically
  - Unfortunately common in distributed systems
  - **Heisenbug**: a software defect that seems to change or disappear during debugging
- **Transient** faults occur only once
  - Often the result of physical phenomena
Suppose there is a bug in one of your CS 361 projects that is a result of improper synchronization, causing you to fail one of the automated tests. However, it does not reproduce in gdb. What kind of fault is this?

- A. Permanent
- B. Intermittent
- C. Transient
Suppose your roommate trips and falls, accidentally hitting the switch on your surge protector and causing your desktop to lose power. What kind of fault is this?

- A. Permanent
- B. Intermittent
- C. Transient
Bit errors

- **Bit error**: low-level fault where a bit is read/written incorrectly
- **Single-bit vs. double-bit vs. multi-bit**
  - Single-Bit Error (**SBE**), Double-Bit Error (**DBE**)
  - **Hamming distance**: # of bits different
- **Potential DRAM source**: "weak bits" in hardware
  - Electrical charge stored in a memory cell capacitor
  - **Critical charge** ($Q_{\text{crit}}$) is the threshold between 0 and 1 values
  - Refreshed often, but sometimes still read incorrectly
- **Radiation and cosmic rays**
Example: GPU fault study

The Titan supercomputer has 18,688 GPUs

Figure 3: Radiation test setup inside the ICE House II, Los Alamos Neutron Science Center (LANSCE), LANL. A similar setup was used at ISIS, Didcot, UK.

Dealing with failure

• **Detection**: discovering failures
  – Active (**pinging**) vs. passive (**wait for messages**)  
  – Issue: unreliability of **timeouts**

• **Prevention**: eliminate the possibility of failure
  – Not possible in a distributed system

• **Avoidance**: divert around failure possibilities
  – Only possible in particular circumstances

• **Recovery**: restore valid system state after a failure
  – **Forward error correction** includes additional info for recovery
Detection and avoidance

- Data-centric
  - Redundancy, diversity, and replication
    - E.g., dual modular redundancy (DMR), TMR
  - Parity bits, checksums, and hashes
    - E.g., cyclic redundancy check (CRC), MD5, SHA

- Computation-centric
  - Acknowledgement (ACK)-based protocols
  - Consensus and voting protocols
    - One-phase vs. two-phase (e.g., Paxos)
Detection and avoidance

• How many total bits must be transmitted to **detect** a single-bit error?
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5
Detection and avoidance

• How many total bits must be transmitted to detect a double-bit error?
  
  – A. 1
  – B. 2
  – C. 3
  – D. 4
  – E. 5
How many total bits must be transmitted to **correct** a single-bit error?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5
Recovery in hardware

- **Hardware (general space vs. safety tradeoff)**
  - **Dual modular redundancy** (DMR) can **detect** a single-bit error
  - **Triple modular redundancy** (TMR) can **recover** one corrupted bit
    - Or detect a double-bit error
  - **Parity bits**
    - *Even* parity bits are 0 if the # of 1s is even; 1 otherwise
      - Special case of CRC (polynomial is $x+1$)
    - *Odd* parity bits are 1 if the # of 1s is even; 0 otherwise

**DMR:**

- 0 0 0 ok (value = 0)
- 0 1 SBE
- 1 0 SBE
- 1 1 ok (value = 1)

**TMR:**

- 0 0 0 ok (value = 0)
- 0 0 1 SBE (value = 0) or DBE
- 0 1 0 SBE (value = 0) or DBE
- 0 1 1 SBE (value = 1) or DBE
- 1 0 0 SBE (value = 0) or DBE
- 1 0 1 SBE (value = 1) or DBE
- 1 1 0 SBE (value = 1) or DBE
- 1 1 1 ok (value = 1)
Recovery codes

- **Hamming** codes (often used in ECC memory) use parity bits
  - Bit position $2^i$ is a parity covering all bits with the $(i+1)$th least significant bit set
  - Each bit is covered by a unique set of parity bits
  - Error locations are identified by summing the positions of the faulty parity bits
  - Can detect & recover SBEs (can be extended to detect DBEs)

- **Reed-Solomon** codes are more complex (but widely used)
  - Function values or coefficients of a polynomial

---

**Hamming code**: parity bits and corresponding data bits

from https://en.wikipedia.org/wiki/Hamming_code
Recovery codes

- QR codes provide multiple recovery % options
  - Four levels: L (7%), M (15%), Q (25%), H (30%)
Recovery in software

- **Software level**
  - **Log**: record of operations (can enable recovery)
  - **Checkpoint**: snapshot of current state
    - Independent vs. coordinated checkpointing
    - Standalone vs. incremental checkpointing
    - Tradeoff: space vs. time (how much to save?)
  - **Restore**: revert system state to a checkpoint
    - May require replaying some calculations
    - Can a checkpoint be restored on a different system?
      - If so, how?