

## 6.033: Fault Tolerance: Reliability via Replication

### Lecture 14

Katrina LaCurts, lacurts@mit.edu

#### 0. Introduction

- Done with OSes, networking
- Now: how to systematically deal with failures, or build "fault-tolerant" systems
  - We'll allow more complicated failures and also try to recover from failures
- Thinking about large, distributed systems. 100s, 1000s, even more machines, potentially logated across the globe.
- Will also have to think about what these applications are doing, what they need

#### 1. Building fault-tolerant systems

- General approach:
  1. Indentify possible faults (software, hardware, design, operation, environment, ...)
  2. Detect and contain
  3. Handle the fault
    - do nothing, fail-fast (detect and report to next higher-level), fail-stop (detect and stop), mask, ...
- Caveats
  - Components are always unreliable. We aim to build a reliable system out of them, but our guarantees will be probabilistic
  - Reliability comes at a cost; always a tradeoff. Common tradeoff is reliability vs. simplicity.
  - All of this is tricky. It's easy to miss some possible faults in step 1, e.g. Hence, we iterate.
  - We'll have to rely on \*some\* code to work correctly. In practice, there is only a small portion of mission-critical code. We have stringent development processes for those components.

#### 2. Quantifying reliability

- Goal: increase availability
- Metrics:
  - MTTF = mean time to failure
  - MTTR = mean time to repair
  - $\text{availability} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$
- Example: Suppose my OS crashes once every month, and takes 10 minutes to recover.
  - MTTF = 30 days = 720 hours = 43,200 minutes
  - MTTR = 10 minutes
  - $\text{availability} = 43,200 / 43,210 = .9997$

#### 3. Reliability via Replication

- To improve reliability, add redundancy
- One way to add redundancy: replication

- Today: replication within a single machine to deal with disk failures
- Tomorrow in recitation: replication across machines to deal with machine failures.

#### 4. Dealing with disk failures

- Why disks?
  - Starting from single machine because we want to improve reliability there first before we move to multiple machines
  - Disks in particular because if disk fails, your data is gone. Can replace other components like CPU easily. Cost of disk failure is high.
- Are disk failures frequent?
  - Manufacturers claim MTBF is 700K+ hours, which is bogus.
  - Likely: Ran 1000 disks for 3000 hours (125 days) => 3 million hours total, had 4 failures, and concluded: 1 failure every 750,000 hours.
  - But failures aren't memoryless: disk is more likely to fail at beginning of its lifespan and the end than in the middle (see slides)

#### 5. Whole-disk failures

- General scenario: entire disk fails, all data on that disk is lost. What to do? RAID provides a suite of techniques.
- RAID 1: Mirror data across 2 disks.
  - Pro: Can handle single-disk failures
  - Pro: Performance improvement on reads (issue two in parallel), not a terrible performance hit on writes (have to issue two writes, but you can issue them in parallel too)
  - Con: To mirror N disks' worth of data, you need 2N disks
- RAID 4: With N disks, add an additional parity disk. Sector i on the parity disk is the XOR of all of the sector i's from the data disk.
  - Pro: Can handle single-disk failures (if one disk fails, xor the other disks to recover its data)
    - Can use same technique to recover from single-sector errors
  - Pro: To store N disks' worth of data we only need N+1 disks
  - Pro: Improved performance if you stripe files across the array. E.g., an N-sector-length file can be stored as one

sector

reads

per disk. Reading the whole file means N parallel 1-sector reads instead of 1 long N-sector read.

- RAID is a system for reliability, but we never forget about performance, and in fact performance influenced much of the design of RAID.
- Con: Every write hits the parity disk.
- RAID 5: Same as RAID 4, except intersperse the parity sectors amongst all N+1 disks to load balance writes. (see slide for diagram)

- You need a way to figure out which disk holds the parity sector for sector  $i$ , but it's done systematically
- RAID 5 used in practice, but falling out in favor of RAID 6, which uses the same techniques but provides protection against two disks failing at the same time.

#### 6. Your future

- RAID, and even replication, don't solve everything.
  - E.g., what about failures that aren't independent?
- Wednesday: we'll introduce transactions, which let us make some abstractions to reason about faults
- Next-week: we'll get transaction-based systems to perform well on a single machine.
- Week after: we'll get everything to work across machines.