CS162
Operating Systems and Systems Programming
Lecture 20

Why Systems Fail and What We Can Do About It

November 18, 2013

Anthony D. Joseph and John Canny
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

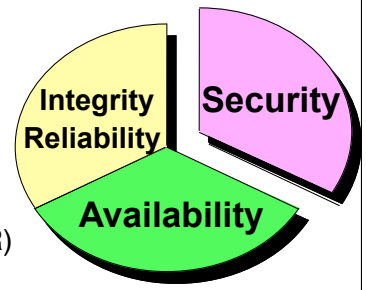
- Definitions for Fault Tolerance
- Causes of system failures
- Fault Tolerance approaches
  - HW- and SW-based Fault Tolerance, Datacenters, Cloud, Geographic diversity

"You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done." **—LESLIE LAMPORT**

Note: Some slides and/or pictures in the following are adapted from slides from a talk given by Jim Gray at UC Berkeley on November 9, 2000.

Dependability: The 3 ITIES

- **Reliability / Integrity:** does the right thing.
  (Need large MTBF)
- **Availability:** does it now.
  (Need small $\frac{\text{MTTR}}{\text{MTBF+MTTR}}$)



- **System Availability:** if 90% of terminals up & 99% of DB up?
  (=> 89% of transactions are serviced on time)

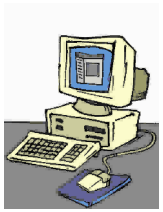
MTBF or MTTF = Mean Time Between (To) Failure
MTTR = Mean Time To Repair

Mean Time to Recovery

- Critical time as further failures can occur during recovery
- Total Outage duration (MTTR) =

Time to Detect	(need good monitoring)
+ Time to Diagnose	(need good docs/ops, best practices)
+ Time to Decide	(need good org/leader, best practices)
+ Time to Act	(need good execution!)

Fault Tolerance versus Disaster Tolerance



- **Fault-Tolerance:** mask local faults
  - Redundant HW or SW
  - RAID disks
  - Uninterruptible Power Supplies
  - Cluster Failover
- **Disaster Tolerance:** masks site failures
  - Protects against fire, flood, sabotage,..
  - Redundant system and service at remote site(s)
  - Use design diversity

High Availability System Classes

Availability %	Downtime per year	Downtime per month	Downtime per week
90% (“one nine”)			
99% (“two nines”)			
99.9% (“three nines”)			
99.99% (“four nines”)			
99.999% (“five nines”)			
99.9999% (“six nines”)			

GOAL: Class 6

Gmail, Hosted Exchange target 3 nines (unscheduled)

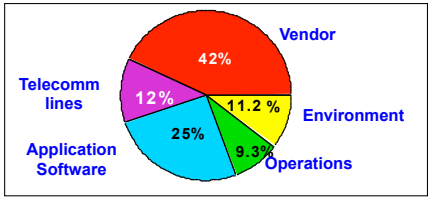
2010: Gmail (99.984), Exchange (>99.9)

UnAvailability ~ MTTR/MTBF

Can cut it by reducing MTTR *or* increasing MTBF

Case Study – Japan

“Survey on Computer Security”, Japan Info Dev Corp., March 1986. (trans: Eiichi Watanabe).



Vendor (hardware and software)	5	Months
Application software	9	Months
Communications lines	1.5	Years
Operations	2	Years
Environment	2	Years
	10	**Weeks**

1,383 institutions reported (6/84 - 7/85)

7,517 outages, MTBF ~ 10 weeks, avg duration ~ 90 MINUTES

To Get 10 Year MTBF, Must Attack All These Areas

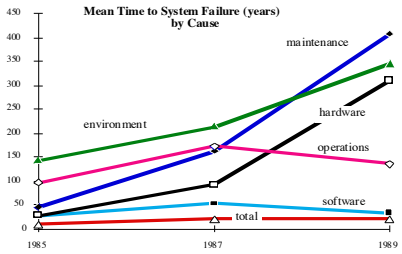
Causal Factors for Unavailability

Lack of best practices for:

- Change control
- Monitoring of the relevant components
- Requirements and procurement
- Operations
- Avoidance of network failures, internal application failures, and external services that fail
- Physical environment, and network redundancy
- Technical solution of backup, and process solution of backup
- Physical location, infrastructure redundancy
- Storage architecture redundancy

Ulrik Franke et al: Availability of enterprise IT systems - an expert-based Bayesian model

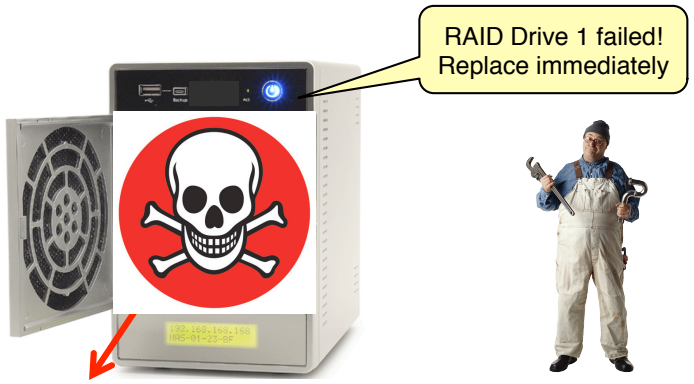
Case Studies - Tandem NonStop Systems
Reported MTBF by Component – *System goal 10yr MTBF*



	1985	1987	1990	
SOFTWARE	2	53	33	Years
HARDWARE	29	91	310	Years
MAINTENANCE	45	162	409	Years
OPERATIONS	99	171	136	Years
ENVIRONMENT	142	214	346	Years
SYSTEM	**8**	**20**	**21**	**Years**

Problem: Systematic Under-reporting

Operations Failures



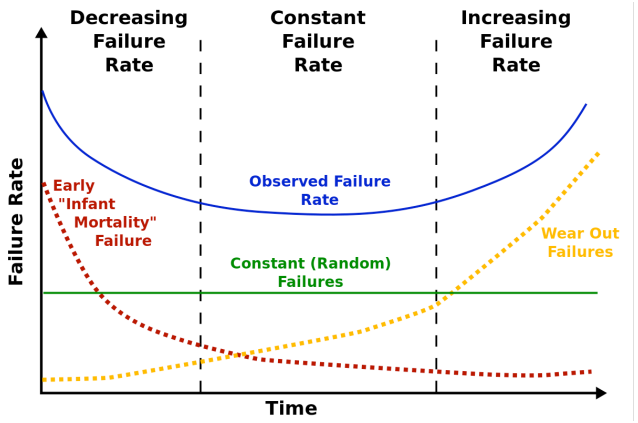
What went wrong??

Operations Failures

Fault Model

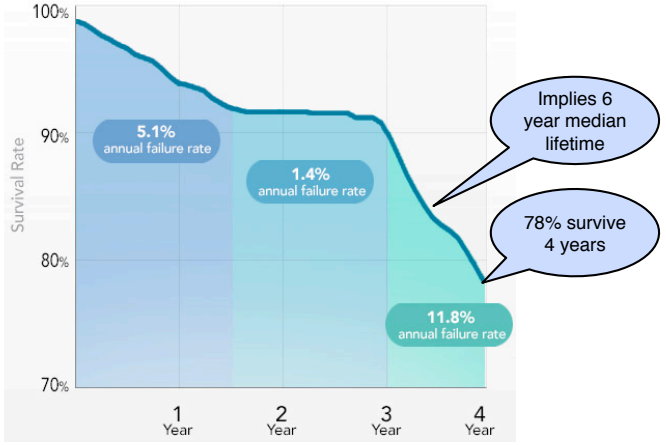
- Assume failures are independent*
  So, single fault tolerance is a big win
- Hardware fails fast (blue-screen, panic, …)
- Software fails-fast (or stops responding/hangs)
- Software often repaired by reboot:
  - Heisenbugs – Works On Retry
  - (Bohrbugs – Faults Again On Retry)
- Operations tasks: major source of outage
  - Utility operations – UPS/generator maintenance
  - Software upgrades, configuration changes

Hardware Failure Rate



https://en.wikipedia.org/wiki/Bathtub_curve

Consumer Disk Drive Failure Rates



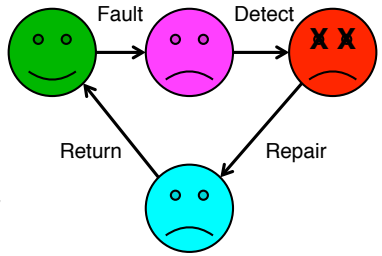
http://blog.backblaze.com/2013/11/12/how-long-do-disk-drives-last/

Traditional Fault Tolerance Techniques

- Fail fast modules: work or stop
- Spare modules: yield instant repair time
- Process/Server pairs: Mask HW and SW faults
- Transactions: yields ACID semantics (simple fault model)

Fail-Fast is Good, but Repair is Needed

Lifecycle of a module
fail-fast gives
short fault latency



High Availability
is low UN-Availability

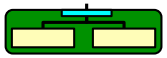
$$\text{Unavailability} \sim \frac{\text{MTTR}}{\text{MTBF}}$$

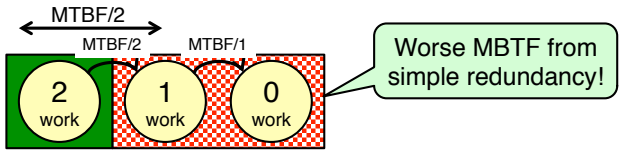
Improving either MTTR or MTBF gives benefit

Simple redundancy does not help much (can actually hurt!)

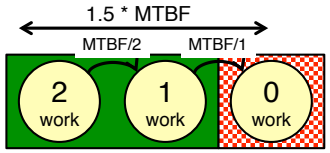
Hardware Reliability/Availability
(how to make HW fail fast)

Duplex module with output comparator:





Fail-Fast: fail if either fails (e.g., duplexed CPUs)



The Airplane Rule:

A two-engine airplane has twice as many engine problems as a one engine plane

Fail-Soft: fail if both fail (e.g., disc, network,...)

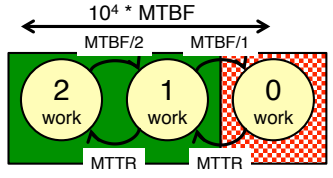
Add Repair: Gain 10^4 Improvement

Adding repair puts module back into service after a failure

Duplex Fail Soft + Repair

Equation: $MTBF^2/(2 * MTTR)$

Yields > 10^4 year MTBF



1 Year MTBF modules
12-hour MTTR

Administrivia

- Project 3 code due Thursday 11/21 by 11:59PM
- Project 4 design due date changed
  - Monday 12/2 by 11:59PM

5min Break

Software Techniques: Learning from Hardware

- Fault avoidance starts with a *good and correct design*
- After that – Software Fault Tolerance Techniques:

  Modularity (isolation, fault containment)

  Programming for Failures: Programming paradigms that assume failures are common and hide them

  Defensive Programming: Check parameters and data

  N-Version Programming: N-different implementations

  Auditors: Check data structures in background

  Transactions: to clean up state after a failure

Try&Catch Alone isn't Fault Tolerance!

```
String filename = "/nosuchdir/myfilename";

try {
     // Create the file
     new File(filename).createNewFile();
}
catch (IOException e) {
     // Print out the exception that occurred
     System.out.println("Unable to create
file ("+filename+"): "+e.getMessage());
}
```

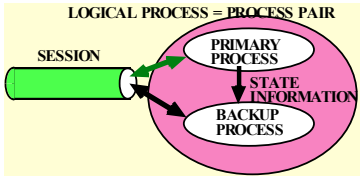
- Fail-Fast, but is this the desired behavior?
- Alternative behavior: (re)-create missing directory?

Fail-Fast and High-Availability Execution

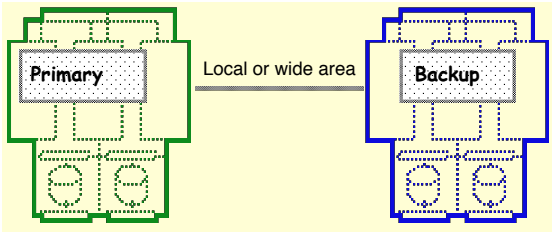
Process Pairs: Instant repair

Use Defensive programming to make a process fail-fast
Have separate backup process ready to "take over" if primary faults

- SW fault is a Bohrbug ➔ *no repair*
  "wait for the next release" or "get an emergency bug fix" or "get a new vendor"
- SW fault is a Heisenbug ➔ *restart process*
  "reboot and retry"
- Yields millisecond repair times
- Tolerates some HW faults

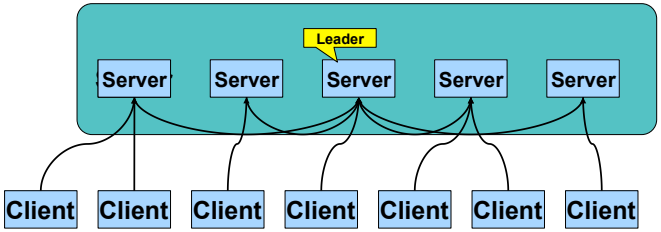


Server System Pairs for High Availability



- Programs, Data, Processes Replicated at 2+ sites
  - Logical System Pair looks like a single system
  - Backup receives transaction log
- If primary fails or operator switches, backup offers service
- *What about workloads requiring more than one server?*

Apache ZooKeeper



- Multiple servers require coordination
  - Leader Election, Group Membership, Work Queues, Data Sharding, Event Notifications, Config, and Cluster Mgmt
- Highly available, scalable, distributed coordination kernel
  - Ordered updates and strong persistence guarantees
  - Conditional updates (version), Watches for data changes

Datacenter is new "server"

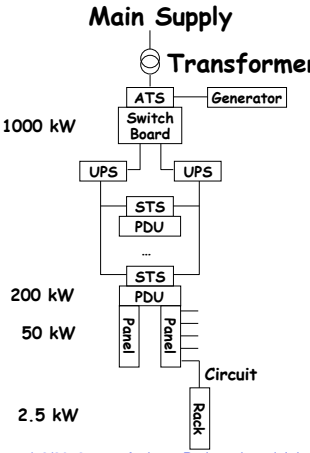
- *What about even larger scale?*
- *"Program"* == Web search, email, map/GIS, …
- *"Computer"* == 1000's computers, storage, network
- Warehouse-sized facilities and workloads
- *Built from less reliable components than traditional datacenters*

photos: Sun Microsystems & datacenterknowledge.com

Datacenter Architectures

- Major engineering design challenges in building datacenters
  - One of Google's biggest secrets and challenges
  - Facebook creating open solution: http://www.facebook.com/note.php?note_id=10150148003778920
  - Very hard to get everything correct!
- Example: AT&T Miami, Florida Tier 1 datacenter
  - Minimum N+1 redundancy factor on all critical infrastructure systems (power, comms, cooling, …)
  - Redundant communications: dual uplinks to AT&T global backbone
  - What about power?

Typical Tier-2 One Megawatt Datacenter



- Reliable power: Mains + Generator, Dual UPS
- Units of Aggregation
  - Rack (10-80 nodes) → PDU (20-60 racks) → Facility/ Datacenter
- Power is 40% of DC costs
- Over 20% of entire DC costs is in power redundancy

X. Fan, W-D Weber, L. Barroso, "Power Provisioning for a Warehouse-sized Computer," ISCA'07, San Diego, (June 2007).

AT&T Internet Data Center Power

Commercial Power Feeds

2 commercial feeds, Each at 13,800VAC
Located near Substation supplied from 2 different grids

AT&T Internet Data Center Power

- Paralleling switch gear
- Automatically powers up all generators when Commercial power is interrupted for more than 7 seconds
  - Generators are shed to cover load as needed
  - Typical transition takes less than 60 seconds
- Manual override available to ensure continuity if automatic start-up should fail

Emergency Power Switchover

AT&T Internet Data Center Power

Back-up Power – Generators and Diesel Fuel

- Four (4) 2,500 kw Diesel Generators Providing Standby Power, capable of producing 10 MW of power
- Two (2) 33,000 Gallon Aboveground Diesel Fuel Storage Tanks

AT&T Internet Data Center Power

- UPS consisting of four battery strings
- Battery strings contain flooded cell Lead Acid batteries
- A minimum of 15 minutes of battery backup available at Full load
- Remote status monitoring of battery strings

UPS Batteries

AT&T Internet Data Center Power

- Motor-Generator pairs clean power
- Eliminate spikes, sags, surges, transients, and all other Over/Under voltage And frequency conditions

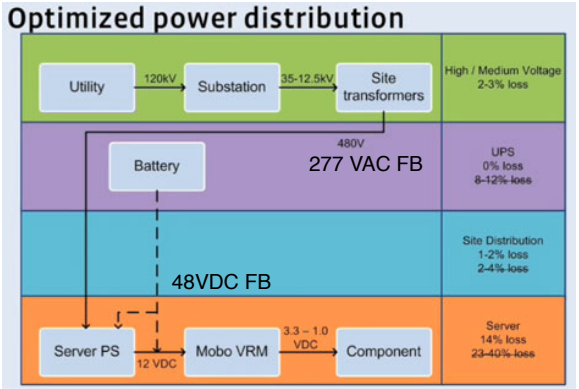
Uninterruptible Power Supply (UPS)

- Four UPS Modules connected in a Ring Bus configuration
- Each Module rated at 1000kVA
- Rotary Type UPS by Piller

AT&T Enterprise Hosting Services briefing 10/29/2008

Alternate Power Distribution Model

- Google and Facebook replaced central UPS with individual batteries*



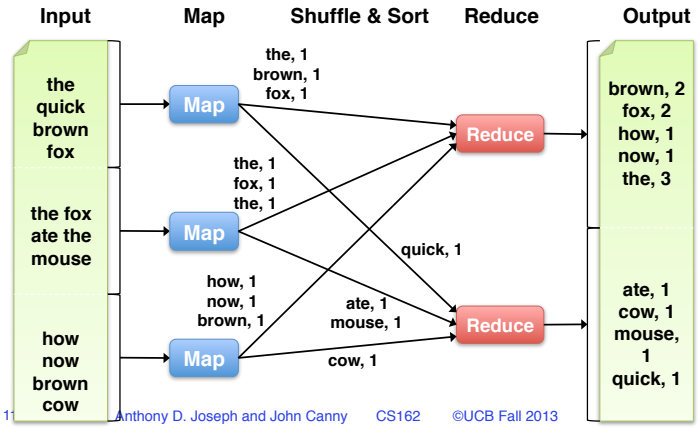
Many Commercial Alternatives

- The rise of Cloud Computing!
- "Inexpensive" virtual machine-based computing resources
  - Instantaneous (minutes) provisioning of replacement computing resources
  - Also highly scalable
- Competition driving down prices
- Easiest way to build a startup!
  - Scale resources/costs as company grows

MapReduce: Programming for Failure

- First widely popular programming model for data-intensive apps on commodity clusters
- Published by Google in 2004
  - Processes 20 PB of data / day
- Popularized by open-source Hadoop project
  - 40,000 nodes at Yahoo!, 70 PB at Facebook
- Programming model
  - Data type: key-value *records*
    - » Map function: (K_{in}, V_{in}) ➔ list(K_{inter}, V_{inter})
    - » Reduce function: $(K_{inter}, \text{list}(V_{inter}))$ ➔ list(K_{out}, V_{out})

Word Count Execution



Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     - » OK for a map because it had no dependencies
     - » OK for reduce because map outputs are on disk
   - If the same task repeatedly fails, fail the job

2. If a node crashes:
   - Relaunch its current tasks on other nodes
   - Relaunch any maps the node previously ran
     - » Necessary because their output files are lost

➢ Tasks must be *deterministic* and *side-effect-free*

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node
   - Take output of whichever copy finishes first

- Critical for performance in large clusters

- *What about other distributed applications?*
  - Web applications, distributed services, …
  - Often complex with many, many moving parts
  - Interdependencies often hidden and/or unclear

Cloud Computing Outages Q4 2012

Vendor	When	Duration	What happened and Why
Microsoft (Xbox Live & Azure)	December 28, 2012 till December 31, 2012	More than 36 Hours	Xbox 360 users were affected after Microsoft's Cloud Save feature broke down on 28 December. The outage continued for the whole weekend, with users unable to access saved games held in the cloud until 31 December. Azure service was also disrupted between 28 and 30 December and Microsoft initially reported that only users of its storage service in the South Central US region were affected. However, it quickly became apparent the outage was also affecting its global Management Portal. The problem, which was blamed on 'faulty nodes' took over 36 hours to resolve in full, with Microsoft issuing an apology "for the interruption and issues it has caused our customers".
Amazon Web Services	December 24, 2012	24 Hours	The outage affected Netflix customers across the United States, Canada and Latin America. It began at 3:30 p.m. Eastern time on Christmas Eve and lasted for some users into Christmas Day. The cause of the failure, was a shutdown of several Elastic Load Balancers (ELB) that distribute network traffic to Netflix customers to support online streaming.
Google Gmail Outage	December 10, 2012	18 Minutes	Gmail was down for 18 minutes last week after a "routine update" briefly broke the e-mail service. The search giant reported that it conducted an update of its load-balancing software from 8:45 a.m. to 9:13 a.m. U.S. West Coast time, and after the problems were detected it managed to quickly roll back the buggy code.
Microsoft Office 365	November 8 & 13, 2012	Nov 8 – 8 hours ; Nov 13 – 5 Hours	An antivirus issue caused the November 8 email issues, according Microsoft blog post. And November 13 outage was due to a combination of maintenance, "network element" and load issues. The post also details steps Microsoft officials said they are taking to prevent these kinds of problems in the future.
Drop Box	October 26, 2012	Several Hours	The interruptions led many to link the issues to an undetermined slowdown of Internet availability, although an exact cause has never been determined.

From: http://analysiscasestudy.blogspot.com/

Introduce Controlled Chaos

- Best way to avoid failure is to fail constantly!
  – John Ciancutti, Netflix
- Inject random failures into cloud by killing VMs
  – Most times, nothing happens
  – Occasional surprises
- April, 2011: EC2 failure brought down Reddit, Foursquare, Quora (and many others)
  – Netflix was unaffected thanks to Chaos Monkey and replication
- Also apply to network and storage systems

http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html

Add Geographic Diversity to *Reduce* Single Points of Failure*

Summary

- Focus on Reliability/Integrity and Availability
  – Also, Security (see next two lectures)
- Use HW/SW FT to increase MTBF and reduce MTTR
  – Build reliable systems from unreliable components
  – Assume the unlikely is likely
  – Leverage Chaos Monkey
- Make operations bulletproof: configuration changes, upgrades, new feature deployment, …
- Apply replication at all levels (including globally)