

CS162
Operating Systems and
Systems Programming
Lecture 20

Why Systems Fail and
What We Can Do About It

April 15, 2013
Anthony D. Joseph
<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 LAMPART

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.

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.2

Dependability: The 3 ITIES

- **Reliability / Integrity:**
does the right thing.
(Need large MTBF)

- **Availability:** does it now.
(Need small MTTR
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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.3

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!)

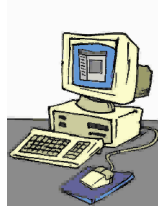
4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.4

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



4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.5

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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.6

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

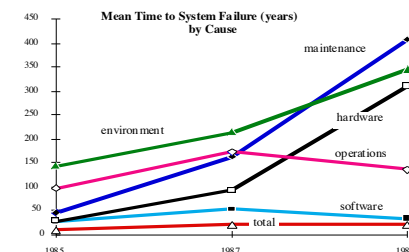
Ulrik Franke et al: Availability of enterprise IT systems - an expert-based Bayesian model

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.7

Case Studies - Tandem Trends Reported MTBF by Component



	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

4/15/2013


Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.8

Operations Failures




RAID Drive 1 failed!
Replace immediately



What went wrong??

4/15/2013
Anthony D. Joseph CS162 ©UCB Spring 2013
Lec 20.9

Operations Failures



RAID Drive 1 failed!
Replace immediately

4
.10

Cloud Computing Outages 2011

Vendor	When	Duration	What Happened & Why
Apple iPhone 4S Siri	November 2011	1 Day	Siri loses even the most basic functionality when Apples servers are down. Because Siri depends on servers to do the heavy computing required for voice recognition, the service is useless without that connection. Network outages caused the disruption according to Apple.
Blackberry outage	October 2011	3 Days	Outage was caused by a hardware failure (core switch failure) that prompted a "triple effect" in RIM's systems. Users in Europe, Middle East, Africa, India, Brazil, China and Argentina initially experienced email and message delays and complete outages and later the outages spread to North America too. Main problem is message backlogs and the downtime produced a huge queue of undelivered messages causing delays and traffic jams.
Google Docs	September 2011	1 Hour	Google Docs word collaboration application cramp, shutting out millions of users from their document lists, documents, drawings and Apps Scripts. Outage was caused by a memory management bug software engineers triggered in a change designed to "improve real time collaboration within the document list.
Windows Live services - Hotmail & SkyDrive	September 2011	2 Hours	Users did not have any data loss during the outage and the interruption was due to an Internet Domain Name Service (DNS). Network traffic balancing tool had an update and the update did not work properly which caused the issue.
Amazon's EC2 cloud &	August 2011	1-2 days	Transformer exploded and caught fire near datacenter that resulted in power outage due to generator failure. Power back up systems at both the data centers failed causing power outages. Transformer explosion was caused by lightning strike but disputed by local utility provider.
Microsoft's BPOS	August 2011	1-2 days	Transformer exploded and caught fire near datacenter that resulted in power outage due to generator failure. Power back up systems at both the data centers failed causing power outages. Transformer explosion was caused by lightning strike but disputed by local utility provider.

From: <http://analysiscasesstudy.blogspot.com/>

4/15/2013
Anthony D. Joseph CS162 ©UCB Spring 2013
Lec 20.11

- ## 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
- 4/15/2013
Anthony D. Joseph CS162 ©UCB Spring 2013
Lec 20.12

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)

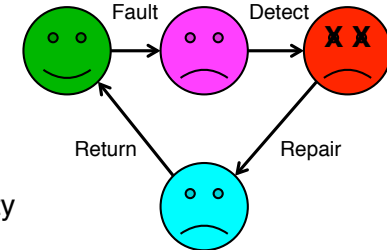
4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.13

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!)

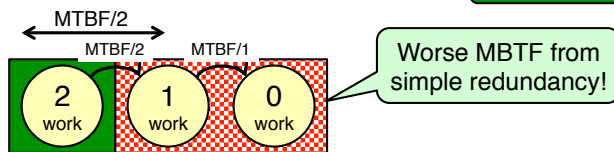
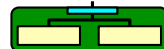
4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

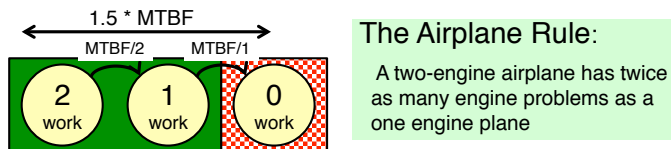
Lec 20.14

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

Duplex module with output comparator:



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



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

4/15/2013

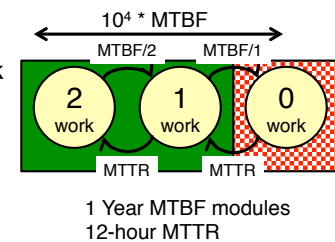
Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.15

Add Repair: Gain 10^4 Improvement

Adding repair puts module back
into service after a failure

Duplex Fail Soft + Repair
Equation: $\text{MTBF}^2 / (2 * \text{MTTR})$
Yields $> 10^4$ year MTBF



1 Year MTBF modules
12-hour MTTR

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.16

5min Break

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.17

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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.18

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?

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

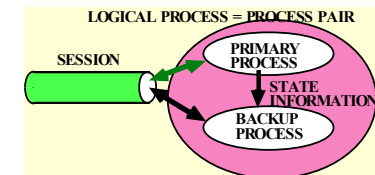
Lec 20.19

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

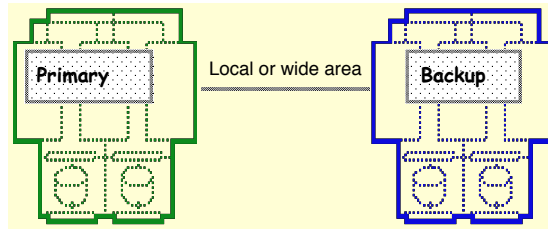


4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.20

Server System Pairs for High Availability



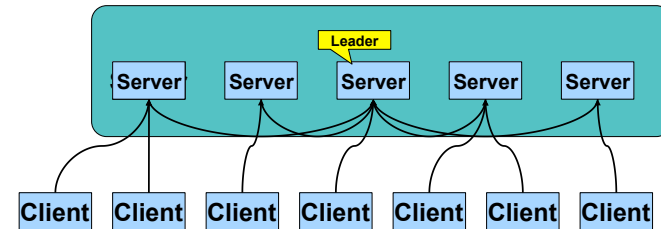
- Programs, Data, Processes Replicated at 2+ sites
 - Logical System Pair looks like a single system
 - Backup receives transaction log
- If primary fails or operator switches, backup offers service
- *What about workloads requiring more than one server?*

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.21

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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.22

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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.23

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



4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.24

MapReduce: Programming for Failure

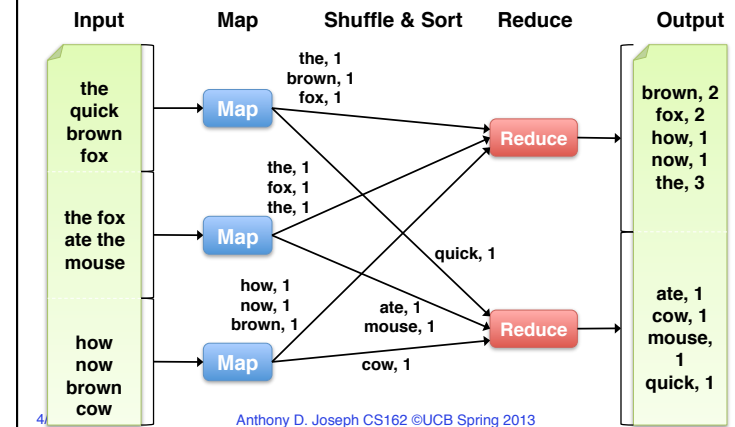
- First widely popular programming model for data-intensive apps on commodity clusters
- Published by Google in 2004
 - Processes 20 PB of data / day
- Popularized by open-source Hadoop project
 - 40,000 nodes at Yahoo!, 70 PB at Facebook
- Programming model
 - Data type: key-value *records*
 - » Map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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.25

Word Count Execution



4

Anthony D. Joseph CS162 ©UCB Spring 2013

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*

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.27

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

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.28

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>

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.29

Add Geographic Diversity to Reduce Single Points of Failure*



4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.30

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)

4/15/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 20.31