ECE 6930-004
HPC Fault Tolerance

BASIC FAULT TOLERANCE CONCEPTS

DR. JON C. CALHOUN

Schadenfreude!

On Tuesday 11 May 2010, Amazon’s EC2 cloud computing service suffered its fourth power outage in a week, with some customers in its US East Region losing service for about an hour. The incident was triggered when a vehicle crashed into a utility pole near one of the company’s data centers, and a transfer switch failed to properly manage the shift from utility power to the facility’s generators.

Lets select a paper

Date	Paper/Topic	Presenter
8/23	Introduction/Syllabus/What is HPC	Calhoun
8/28	Basic Fault Tolerance Concepts	Calhoun
8/30	Toward Exascale Resilience	Calhoun
9/4	Lessons Learned From the Analysis of System Failures at Petascale: The Case of Blue Waters	
9/6	Basics of Checkpoint-restart	Calhoun
9/11	Basics of Checkpoint-restart	Calhoun
9/13	Reasons for a Pessimistic or Optimistic Message Logging Protocol in MPI Uncoordinated Failure Recovery	
9/28	FTI: high performance Fault Tolerance Interface for hybrid systems	
9/20	MCRENGINE: A Scalable Checkpointing System Using Data-Aware Aggregation and Compression	
9/25	What is a soft error?	Calhoun

Outline

Taxonomy of common terms

Modeling reliability

Basics of failure detection in a distributed environment

Definitions

Our definitions come from [Snir et al. 2013]

Based heavily on [Avizenis et al. 2004]
- Considered the canonical definitions in the area (2000+ citations)

Modifications add specific definitions to the HPC domain

This lecture will cover the major terms
- Future lectures will expand our vocabulary

Major definitions

System: an entity that interacts with other entities

Component/subsystem: a system that is part of a larger system

Service: a system’s externally perceived behavior

Total state: a system’s computation, communication, stored information, interconnection, and physical condition

Major definitions

Fault: the cause of an error (e.g., a bug, stuck bit, alpha particle)

Error: the part of total state that may lead to a failure (e.g., a bad value)

Failure: a transition to incorrect service (an event, e.g., the start of an unplanned service outage)

Degraded mode/partial failure: the failure of a subset of services

Fault characteristics

Active: Fault causes an error

Dormant: Fault does not cause an error. The dormant fault is *activated* when it causes an error

Propagation: error moves from one component to another

Permanent: Presence is continuous in time

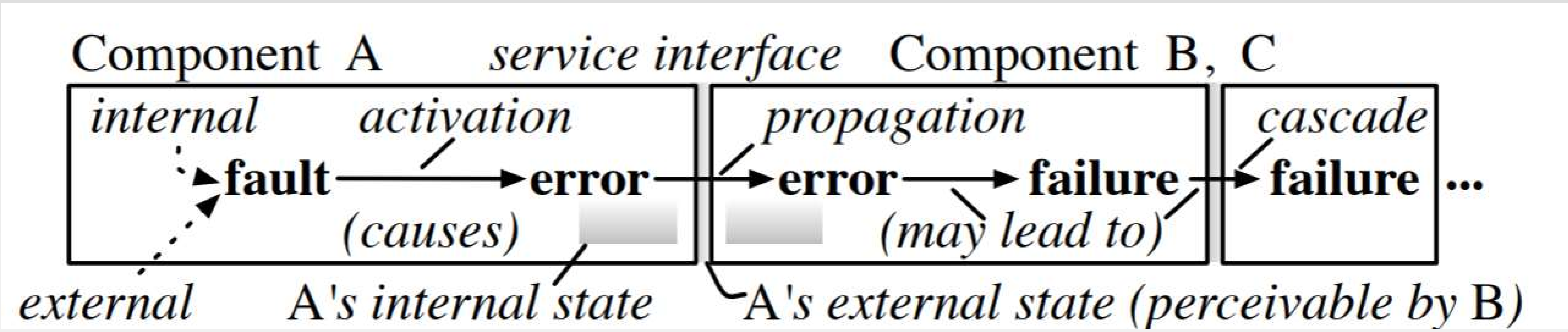
Transient: Presence is temporary

Intermittent: Fault is transient and reappears

Hard/solid: Activation is systematically reproducible

Soft/elusive: Activation is not systematically reproducible

What is the relationship?



Error propagation and cascading failures [Snir et al. 2013]

Error characteristics

Errors indicate the presence of incorrect state in the system

Errors are classified as

- **Detected:** indicated by error message or signal
- **Latent/silent:** not detected
- **Masked:** not causing a failure
- **Soft:** due to a transient fault

What is an error?

What causes errors?

Fault tolerance techniques Detect + =

Error detection: identify the presence of an error

- **Concurrent:** occurs during service delivery
- **Preemptive**: occurs during planned service outage

Error detection informs of the presence of an error, but not the system state that is incorrect

We will discuss papers on soft error detection later in the semester

Fault tolerance techniques Detect + Recover =

Recovery: goal prevent faults from causing failures

- **Error handling:** eliminate errors
  - **Rollback:** revert to previous correct state (e.g., checkpoint, retry)
  - **Rollforward:** move forward to a new correct state
  - **Compensation:** correct the error (e.g., via redundancy)

- **Fault handling:** prevents faults from reactivating
  - **Diagnosis:** identifies fault location and type (e.g., root cause analysis)
  - **Isolation:** excludes from interaction with other components
  - **Reconfiguration:** replaces component or moves work elsewhere
  - **Reinitialization:** performs a pristine reset of state (e.g., reboot)

Several papers cover checkpoint-restart (rollback recovery)

Outline

Taxonomy of common terms

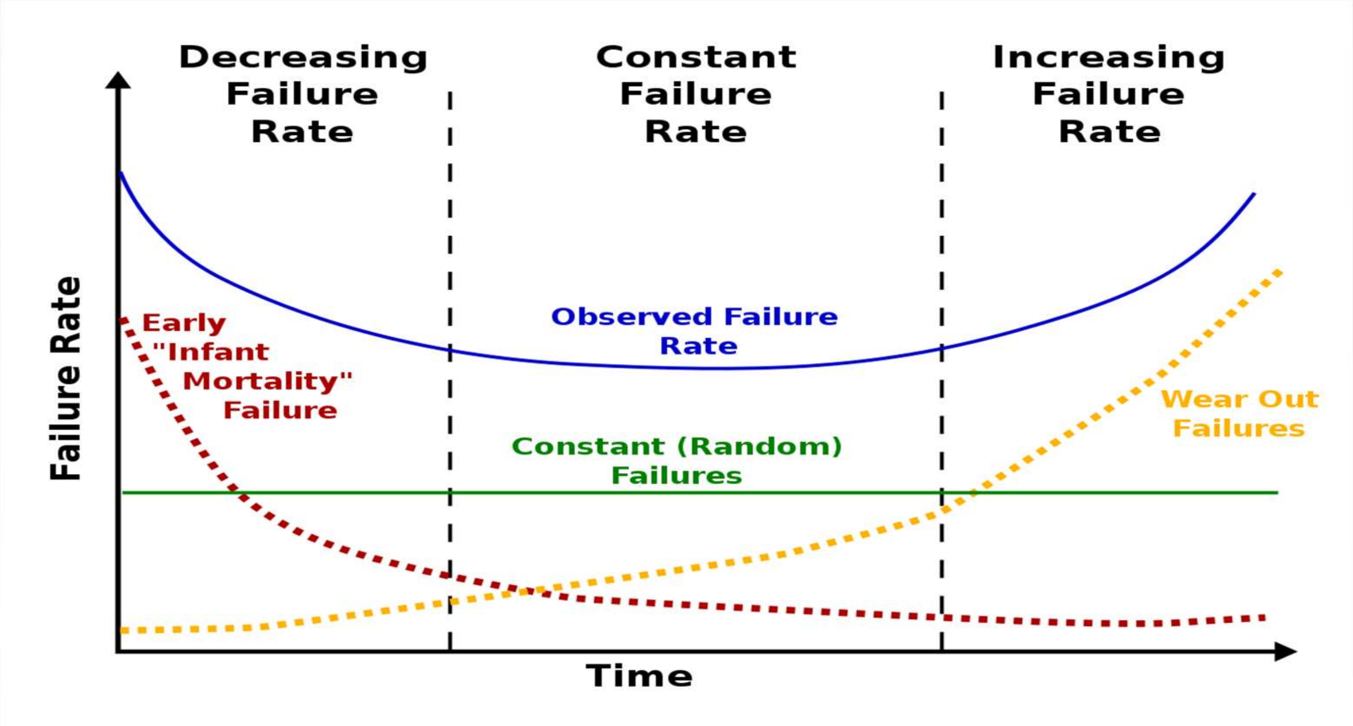
Modeling reliability

Basics of failure detection in a distributed environment

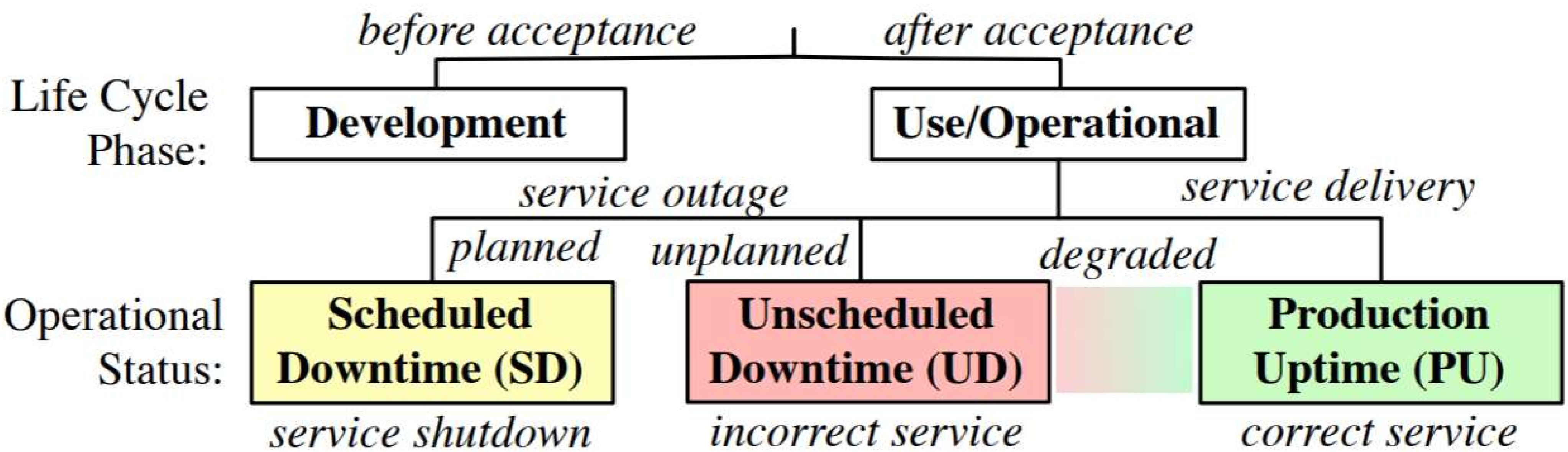
"If you can not measure it, you can not improve it." – Lord Kelvin

Breaking down a system’s lifetime

“Bathtub Curve” of observed failure rate [Wikipedia 2017]



Two modes of operation



System’s operational status [Snir et al. 2013]

Characterizing workload

T_{solve}: idea runtime in a fault-free system

$T_{wallclock}$: actual runtime in a real system

$$Efficiency_{workload} = \frac{T_{solve}}{T_{wallclock}}$$

$$Overhead \; = T_{wallclock} - T_{solve}$$

This metric is an instantons metric and does not evolve overtime

- Overhead now is more important than overhead in the future

Characterizing availability

Overhead impacts how available the system is for normal (useful) operations

Is the system available when:

- X% of compute nodes up
- File system down
- Late notification of unscheduled downtime

What makes a system available?

We won’t parse the particulars

Availability Example

6	1	2	0.3	3.7	1	3	0.4	2.6	2

- System up-time (time when the system functions as specified)
- System scheduled down-time
- System unscheduled down-time

uptime_periods = {6,2,3.7, 3, 2.6}
scheduled_downtime_periods = {1, 1, 2}
unscheduled_downtime_periods = {0.3, 0.4}

uptime = 17.3
scheduled_downtime = 4
unscheduled_downtime = 0.7

num_uptime = 5
num_scheduled_downtime = 3
num_unscheduled_downtime = 2

Availability Example

uptime_periods = {6,2,3.7, 3, 2.6}
scheduled_downtime_periods = {1, 1, 2}
unscheduled_downtime_periods = {0.3, 0.4}

uptime = 17.3
scheduled_downtime = 4
unscheduled_downtime = 0.7

num_uptime = 5
num_scheduled_downtime = 3
num_unscheduled_downtime = 2

$$scheduled\_availability = \frac{total\_time - scheduled\_downtime}{total\_time}$$

$$scheduled\_availability = \frac{22 - 4}{22} = 81.8\%$$

Availability Example

uptime_periods = {6,2,3.7, 3, 2.6}
scheduled_downtime_periods = {1, 1, 2}
unscheduled_downtime_periods = {0.3, 0.4}

uptime = 17.3
scheduled_downtime = 4
unscheduled_downtime = 0.7

num_uptime = 5
num_scheduled_downtime = 3
num_unscheduled_downtime = 2

$$actual\_availability = \frac{up\_time}{total\_time}$$

$$actual\_availability = \frac{17.3}{22} = 78.6\%$$

Availability Example

uptime_periods = {6,2,3.7, 3, 2.6}
scheduled_downtime_periods = {1, 1, 2}
unscheduled_downtime_periods = {0.3, 0.4}

uptime = 17.3
scheduled_downtime = 4
unscheduled_downtime = 0.7

num_uptime = 5
num_scheduled_downtime = 3
num_unscheduled_downtime = 2

$$Mean\ Time\ Between\ Failures\ (MTBF) = \frac{total\_time}{num\_unscheduled\_downtime} = \frac{22}{2} = 11\ days$$

$$Mean\ Time\ To\ Interrupt\ (MTTI) = \frac{uptime}{num\_unscheduled\_downtime} = \frac{17.3}{2} = 8.65\ days$$

Availability Example

uptime_periods = {6,2,3.7, 3, 2.6}
scheduled_downtime_periods = {1, 1, 2}
unscheduled_downtime_periods = {0.3, 0.4}

uptime = 17.3
scheduled_downtime = 4
unscheduled_downtime = 0.7

num_uptime = 5
num_scheduled_downtime = 3
num_unscheduled_downtime = 2

$$Mean\ Time\ To\ Repair\ (MTTR) = \frac{unscheduled\_downtime}{num\_unscheduled\_downtime} = \frac{0.7}{2} = 0.35\ days$$

$$Failures\ In\ Time\ (FIT) = \frac{10^9}{MTBF} = \frac{10^9\ hours}{11\ days} = 3,787,878.79$$

Failures in time (FIT)

FIT expresses the number of failures over the course of 1 billion hours of operation

Does not accurately describe *when* should we expect a failure

Does not easily account for *connected* devices
- CPU + GPU + DRAM + HDD

Let's use statistics to model reliability

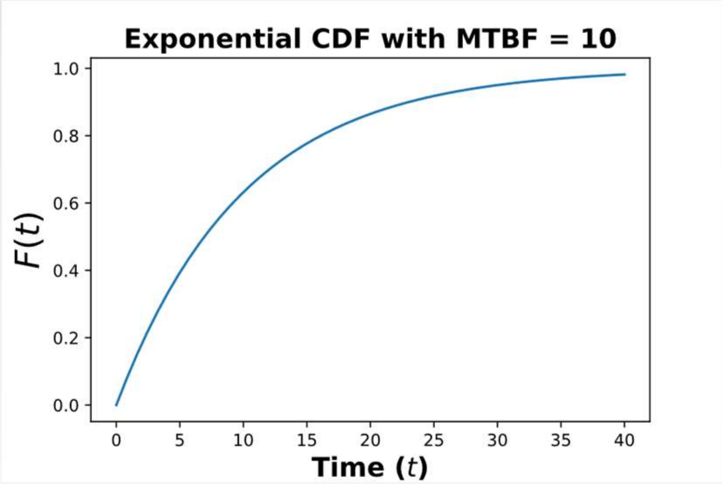
Statistical models

Analysis of failure and recovery algorithms assume that failures occur based on a probabilistic process

- Closed-form description
- Failures are independent
  - Not true(bathtub model, cascading failures)

As time increases, we expect the probability of a failure to increase as well

- **Cumulative Distribution Function** (CDF): Probability of failure before time t



Exponential CDF:

$F(t) = 1 - e^{-\lambda t}$, where λ is the failure rate λ = 1 / MTBF

Weibull CDF:

$F(t) = 1 - e^{(-\lambda t)^k}$, can be used to model

- decreasing failure rate ($k < 1$)
- constant failure rate ($k = 1$)
- increasing failure rate ($k > 1$)

Probability of failing between time and t_1 and t_2 is $\mathrm{F}(t_2) - \mathrm{F}(t_1)$