Introduction to Fault Tolerance

Distributed Systems

Dependability

A **component** provides **services** to **clients**. To provide services, the component may require the services from other components $\Rightarrow$ a component may **depend** on some other component.

Specifically

A component C depends on C^* if the **correctness** of C's behavior depends on the correctness of C^*'s behavior. (Components are processes or channels.)

Requirements related to dependability

Requirement	Description
Availability	Readiness for usage
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Maintainability	How easy can a failed system be repaired

Hardware and software components are never perfect and can fail in different ways.

Reliability Model

- R(t): Probability of no failures till time t
- N: Total number of components — identical
- G(t) : Number of good/available components
- F(t) : Number of bad/failed components

$F(t) + G(t) = N$

1. $R(t) = \frac{G(t)}{N} = 1 - \frac{F(t)}{N}$
2. Failure rate is defined as: $\lambda(t) = \frac{1}{G(t)} \frac{dF(t)}{dt}$ $= \lambda$
3. Differentiating R: $\frac{dR(t)}{dt} = -\frac{1}{N}\frac{dF(t)}{dt}$
4. Assume a constant failure rate $\lambda(t) = \lambda$
5. $\frac{dR(t)}{dt} = -\lambda R(t)$
6. $R(t) = e^{-\lambda t}$

$\frac{dF}{dt} = \lambda \cdot G(t)$

$\frac{dR}{dt} = -\frac{1}{N}\lambda \cdot G(t)$

$\int \frac{dR}{R} = \int -\lambda t$

$\log R = -\lambda t$

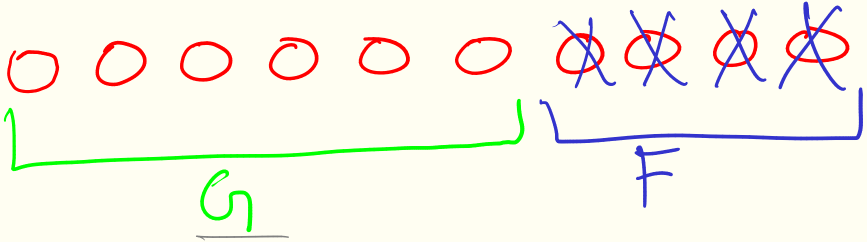
Imagine you want to understand the number of remaining drives from a big batch of hard drives

R(1 year): What is prob of no failures in a 1 year interval.

$N = 10^6$

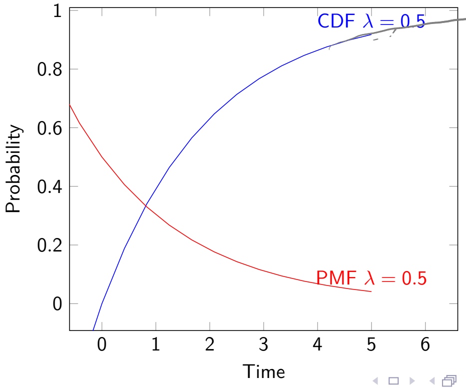
Qs: # failed disks on any given day.

$\approx$ Failure rate $= \lambda(t)$



Exponential Distributions

- $R(t) = e^{-\lambda t}$
- Number of failures till time t, $F(t) = 1 - R(t) = 1 - e^{-\lambda t}$
- This is the CDF of the exponential distribution!
- Probability mass function: $\frac{dF(t)}{dt} = \lambda e^{\lambda t}$



$F(t) = 1 \Rightarrow 1 - e^{-\lambda t} = 1$

$\Rightarrow e^{-\lambda t} = 0$

@ $t \to \infty$ for $\lambda > 0$

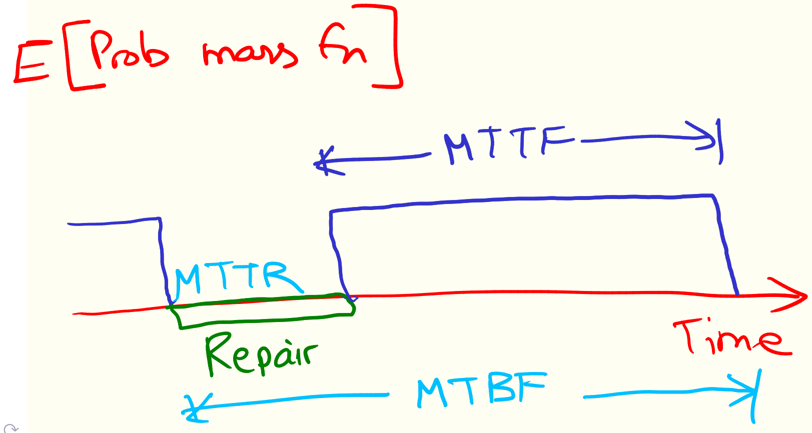
Reliability versus availability

Reliability $R(t)$ of component C

Conditional probability that C has been functioning correctly during $[0, t)$ given C was functioning correctly at time $T = 0$.

Traditional metrics

- **Mean Time To Failure (MTTF)**: The average time until a component fails $= 1/\lambda$
- **Mean Time To Repair (MTTR)**: The average time needed to repair a component.
- **Mean Time Between Failures (MTBF)**: Simply MTTF + MTTR.



Reliability versus availability

Availability $A(t)$ of component C

Average fraction of time that C has been up-and-running in interval $[0, t)$.

- Long-term availability A: $A(\infty)$
- **Note:** $A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF+MTTR}$

Observation

Reliability and **availability** make sense only if we have an accurate notion of what a **failure** actually is.

Failures, errors, and faults

Term	Description	Example
Failure	A component is not living up to its specifications	Crashed program
Error	Part of a component that can lead to a failure	Programming bug
Fault	Cause of an error	Sloppy programmer

Error causes faults that lead to component failures.

Handling Faults

Term	Description	Example
Fault prevention	Prevent the occurrence of a fault	Don't hire sloppy programmers
Fault tolerance	Build a component such that it can mask the occurrence of a fault	Build each component by two independent programmers
Fault removal	Reduce the presence, number, or seriousness of a fault	Get rid of sloppy programmers
Fault forecasting	Estimate current presence, future incidence, and consequences of faults	Estimate how a recruiter is doing when it comes to hiring sloppy programmers

Redundancy

Reliability Modeling.

Failure models

Type	Description of server's behavior
Crash failure	Halts, but is working correctly until it halts
Omission failure	Fails to respond to incoming requests
Receive omission	Fails to receive incoming messages
Send omission	Fails to send messages
Timing failure	Response lies outside a specified time interval
Response failure	Response is incorrect
Value failure	The value of the response is wrong
State-transition failure	Deviates from the correct flow of control
Arbitrary failure	May produce arbitrary responses at arbitrary times

Halting failures

C no longer perceives any activity from C^* — a **halting failure**? Distinguishing between a **crash** or **omission/timing failure** may be impossible.

Asynchronous versus synchronous systems

- **Asynchronous system:** no assumptions about process execution speeds or message delivery times $\rightarrow$ **cannot reliably detect crash failures**.
- **Synchronous system:** process execution speeds and message delivery times are bounded $\rightarrow$ **we can reliably detect omission and timing failures**.
- In practice we have **partially synchronous systems:** most of the time, we can assume the system to be synchronous, yet there is no bound on the time that a system is asynchronous $\rightarrow$ **can normally reliably detect crash failures**.

Halting failures

Assumptions we can make

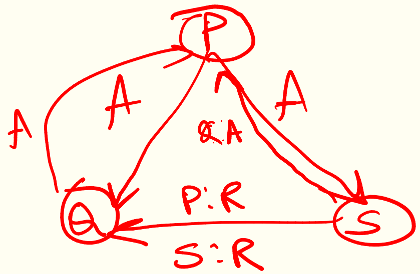
Halting type	Description
Fail-stop	Crash failures, but reliably detectable
Fail-noisy	Crash failures, eventually reliably detectable
Fail-silent	Omission or crash failures: clients cannot tell what went wrong
Fail-safe	Arbitrary, yet benign failures (i.e., they cannot do any harm)
Fail-arbitrary	Arbitrary, with malicious failures

Electronic Door lock.
Fail: Circuit Failure due to high voltage.
Safe State: Locked.

Byzantine Faults

- Presenting different symptoms to different observers
- Most challenging failure mode
- Byzantine Generals Problem: coordinate an attack
- Generals must vote and agree on attack/retreat decision
- Votes are multi-cast (No centralized ballot)
- Trecherous generals can send attack votes to some and retreat to others
- Agreement reached with majority of non-faulty generals
- For n faulty processes, cannot have agreement with only $3n$ processes

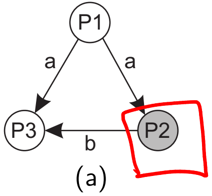
You can tell from the historical allegory that this is Lamport's work...



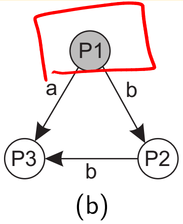
3 Generals with 1 Traitor

Essence

We consider process groups in which communication between process is **inconsistent**: (a) improper forwarding of messages, or (b) telling different things to different processes.



(a)



(b)

Distributed Algorithms With Failures

Exercise

Think about behavior of various distributed algorithms in presence of failures:

- Leader election
- Total Order Multicast