CSCE 5760: Design For Fault Tolerance

Review:
Probability related concepts
Random Variable
Continuous and Discrete RV
Probability Mass or density functions

Discrete: Probability mass function associates the probability with a specific value
Example: consider throwing a pair of dice
\[ \text{Prob}(X=12) = ? \]

Continuous: Probability density function associates probability with a small range of values.
For example: The average temp in Denton on Aug 15
\[ \text{Prob (Temp: } = 100 \pm 0.00001) = \]

Examples of discrete distributions
Bernoulli, Binomial
Geometric, Hypergeometric
Poisson, Multinomial

Examples of Continuous distributions:
Normal, Exponential, Weibul,

Stochastic Variable?
A random variable that depends on some other dimension
time: the value at a specific time
area: the value on 2/3 dimensional space
(temp at a specific pixel)

We can also think of multiple but identical RV simultaneously operating across time or other domains
CSCE 5760: Design For Fault Tolerance

Reliability: Probability that a component (or a system) did not fail during a specified interval \([0,1]\) given that the component is working at time \(\text{time} = 0\)

\[
1 - \int_0^t f(t) dt = 1 - F(t)
\]

Here: \(f(t)\) is the probability density function of failure behavior of the component

\(F(t)\) is called the cumulative distribution: the sum of all the probability failure that the component fails at time \(t\) or before

Mean or average also expected value

Discrete random variable

\[
E[x] = \sum x \cdot f(x)
\]

Continuous distribution:

\[
E[x] = \int x \cdot f(x) dt
\]

CSCE 5760: Design For Fault Tolerance

What is standard deviation

- we can define expected values as central moments
- then we can talk about higher order moments

Second order central moment or variance is given by

\[
V(x) = \sum (x - E(x))^2 f(x) \quad V(x) = \int (x - E(x))^2 * f(x) dt
\]

We can consider 3\(^{rd}\), 4\(^{th}\),… central movement

\((x-E(x))^3, \ldots\)

Variance: describes how values are distributed between min and max

3\(^{rd}\) Central moment is called skewness: How values around mean are distributed more value to the left or right of mean

4\(^{th}\): Kurtosis: How flat the distribution near the mean
Constant Failure rate and Exponential Distribution

If the module has a failure rate which is constant over time –

\[ \lambda(t) = \lambda \]
\[ \frac{dR(t)}{dt} = -\lambda R(t) \quad ; R(0)=1 \]

The solution of this differential equation is

\[ R(t) = e^{-\lambda t} \]
\[ f(t) = \lambda e^{-\lambda t} \]
\[ F(t) = 1 - e^{-\lambda t} \]

A module has a constant failure rate if and only if \( T \), the lifetime of the module, has an exponential distribution

---

Failure rate or Hazard rate (instantaneous failure rate)

\[ z(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - F(t)} \]

It can be shown that the above definition of \( z(t) \) is the conditional probability that the next failure occurs in \((t_1, t_1+dt)\) – very small interval given that the last failure occurred at \( t_1 \).

Note that \( \frac{dR(t)}{dt} = -f(t) \)

For exponential, Hazard rate (or instantaneous failure rate) becomes \( 1/\lambda \).

How do we define MTTF?
CSCE 5760: Design For Fault Tolerance

Mean Time to Failure (MTTF)

MTTF - expected value of the lifetime $T$

$$MTTF = E[T] = \int_0^\infty t \cdot f(t) dt$$

$$\frac{dR(t)}{dt} = -f(t)$$

$$MTTF = -\int_0^\infty \frac{dR(t)}{dt} \cdot dt = \int_0^\infty R(t) dt$$

$R(t) = 0$ if $t = 0$, and $t = \infty$

If the failure rate is a constant $\lambda$

$$R(t) = e^{-\lambda t}$$

$$MTTF = \int_0^\infty t \cdot \lambda e^{-\lambda t} dt = \int_0^\infty e^{-\lambda t} dt = \frac{1}{\lambda}$$

Notice how MTTF is related to failure rate

CSCE 5760: Design For Fault Tolerance

Weibull Distribution – used to represent bathtub curve

Most calculations of reliability assume that a module has a constant failure rate $\lambda$ (or equivalently - an exponential distribution for the module lifetime $T$)

There are cases in which this simplifying assumption is inappropriate

Example - during the ‘婴儿 mortality’ and ‘wear-out’ phases of the bathtub curve

Weibull distribution for the lifetime $T$ can be used instead

We can describe the Bathtub curve for failure rates of electronic components

The Weibull distribution has two parameters, $\lambda$ and $\beta$

The density function of the component lifetime $T$:

$$f(t) = \lambda \beta t^{\beta-1} e^{-\lambda t^\beta}$$

The failure rate for the Weibull distribution is

$$\lambda(t) = \lambda \beta t^{\beta-1}$$

$\lambda(t)$ is decreasing with time for $\beta < 1$, increasing with time for $\beta > 1$, constant for
CSCE 5760: Design For Fault Tolerance

Weibull distribution - Equation

Reliability for Weibull distribution is
\[ R(t) = e^{-\lambda t^\beta} \]

MTTF for Weibull distribution is
\[ MTTF = \Gamma \left( \frac{1}{\beta} \right) \left( \frac{1}{\beta^{1/\beta}} \right) \]

(\( \Gamma(x) \) is the Gamma function
\[ \Gamma(x) = \int_0^\infty y^{x-1} e^{-y} \, dy \]
\[ \Gamma(x>1) = (x-1)\Gamma(x-1) \]
\[ \Gamma(x=0) = \Gamma(1) = 1 \]
\[ \Gamma(n) = (n-1)! \]

Note the recursive nature of the Gamma function.

The special case \( \beta = 1 \) is the exponential distribution with a constant failure rate \( \lambda \).

CSCE5760 September 03, 2019

CSCE 5760: Design For Fault Tolerance

Weibull distribution - Example

Exercise 2.3.

To get a feel for the failure rates associated with the Weibull distribution, plot them for the following parameter values as a function of the time, \( t \):
(a) Fix \( \lambda = 1 \) and plot the failure rate curves for \( \beta = 0.5; 1.0; 1.5 \).
(b) Fix \( \beta = 1.5 \) and plot the failure rate curves for \( \lambda = 1; 2; 5 \).

Solution:
The failure rate for the Weibull distribution is
\[ \lambda(t) = \lambda t^{\beta-1} \]

If \( \beta > 1 \), failure rate increases with time
If \( \beta < 1 \), failure rate decreases with time
CSCE 5760: Design For Fault Tolerance

Availability
Availability, A(t) – fraction of the time the system is UP during the interval [0,t]

We may also be interested in: probability that the system is available at time t
point availability Ap(t)
Availability of the system in the long run, A

\[ A = \lim_{t \to \infty} A(t) = \lim_{t \to \infty} Ap(t) \]

Availability assumes repair
\[ A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \]

We are defining a new term Mean-Time-Between-Failures as MTTF+MTTR
Availability depends how fast we can repair
Repair may involve just replacing failed component with good one

Example: MTTF is 1 hour, MTTR = 1 sec \( A = \frac{3600}{3601} = 99.97\% \)

CSCE 5760: Design For Fault Tolerance

How to develop failure models?
Modeling hardware failures can be based on
models of physical behaviors
how current, temp., EM impacts
models of manufacturing process
maturity models
Verified by testing
how good is the testing?

What about software failures?
Report all faults detected during testing
Use historical data on faults found to estimate expected number of faults
Use these estimates to model "residual" faults

Same ideas with maintenance or upgrade processes
CSCE 5760: Design For Fault Tolerance

Let us understand origins of faults.

specification mistakes
– incorrect algorithms, incorrectly specified requirements
  (timing, power, environmental)
implementation mistakes
– poor design, software coding mistakes
component defects
– manufacturing imperfections, random device defects,
  components wear-outs
external factors
– radiation, lightning, operator mistakes

Cause-and-effect relationship

At what level should we model failures?

Starting from lowest level – circuit or device level
  to system level (including networking etc)

Device or circuit level:
  Oxide breakdown – large electric fields in insulator or gate oxide
  Electromigration – drifting of metal atoms towards the cathode
    wear phenomenon
  Hot electron – trapped in gate oxide due to high temperature
  transient fault – produced by alpha particles or radiation

  some are due to manufacturing and some due to operating conditions
CSCE 5760: Design For Fault Tolerance

Switching circuit level faults
- Stuck-at faults – stuck at 1 or stuck at 0
- Bridging – two or more signal lines are shorted
- Group or unidirectional fault
  - multiple lines are stuck in the same direction

Studies show that majority of faults are stuck at
But with miniaturization bridging faults may become more serious

Logic Level faults at higher levels
- Stuck at faults at logic gate level
  - NAND, NOR etc

System or microarchitecture level
- registers
- control units, or specific pipelines
- ALU, memory etc
- Buses or other interconnection networks

Even more abstract
- Cores, caches, network on chip
- PC board level failures
CSCE 5760: Design For Fault Tolerance

Some thoughts about modeling networks: modeled basically as graphs

What can fail in networks?
Nodes, Arcs

Nodes and edges
  Redundancy $\Rightarrow$ multiple paths
  Arc and Node connectivity

Other metrics (studied in parallel processing)
  diameter, bisection bandwidth

A few other assumptions about the nature of faults and how to model/measure

A common-mode fault is a fault which occur simultaneously in two or more redundant components
  Caused by phenomena that create dependencies between components
    - common communication bus
    - shared environmental conditions
    - common source of power
    - design mistake

Design diversity is the implementation of one or more variant of the redundant component

Fault Models

It is very difficult to analyze a system without assuming some fault models
  - hard to design test procedures
  - hard to simulate faults

To make the problem more manageable, we need to restrict our attention to a subset of all faults what can occur
Fault Models

Fault model is a logical abstraction describing the functional effect of physical defect.

Models depend on the level (hardware component, device, network, software) we are dealing with:
- component, gate level, circuit, logic, transistor, layout

We may rely on different types of models:
- stuck-at, transition, coupling

Models depend on how we can test and measure faults:
- **Hardware Level → Logic models**

  most commonly used model → **stuck at faults**

  - the effect of the fault is modeled by having a line in the circuit permanently fixed to 0 or 1 value
  - the basic functionality of the circuit is not changed
    - gates remain the same
    - combinational circuit is not transformed to sequential

Testing for Hardware Logic faults

How do we test if we have a stuck-at (either zero or 1) fault?

We use an input such that the actual output does not match expected result.

How many inputs should we use?

**Complete test set** is a set of tests detecting all faults in the circuit (of a specified type)

**Minimal complete test set** is a complete test set with the minimal number of tests

*Note minimal set is normally smaller than complete set*

Let us consider how we can use truth tables to find which input combinations allow us to discover stuck at faults.
Truth-table based method for finding tests for stuck-at faults

To find tests for some stuck-at fault $\alpha$:

- Write truth tables for the function without fault, $f$, and the function with fault, $f^\alpha$.
- All input assignments of the truth table for which $f \neq f^\alpha$ are tests for the fault $\alpha$.

**Example**

Let $f = x_1x_2 + x_2x_3$ and $f^\alpha = x_2x_3$.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
<th>$f^\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Here we are testing for Line 1 stuck-at-zero.

We find that input [1,1,0] reveals the stuck at fault. None of the other input combinations reveal the fault.

If we want to extend this and find all possible stuck-at-zero/one for every line, we may have to test for may combinations.

Here we have 3 inputs, each can take 0/1, so we have only 8 combinations. But most circuits have more inputs.

---

We need to find a subset of all input combinations but still try to find all possible faults.

**Fault Coverage:** How many faults are covered (tested) by an input set.

**Complete minimal test set** is the minimum number of inputs that test for all possible faults.

**Example**

Let's say we have a circuit with states $s = 0, 1, 2, 3$ and transitions $s \rightarrow 0, 1, 2, 3$.

<table>
<thead>
<tr>
<th>$s_{old}$</th>
<th>$s_{new}$</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s-a-0</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>s-a-1</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s-a-0</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s-a-1</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>s-a-0</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>s-a-1</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The complete minimal test set is $\{(00), (01), (10)\}$.

There are algorithms and tools to generate such minimal test sets. Specify circuit using logical expressions.

How about sequential circuits?
Software Faults

Software differs from hardware in several aspects:
– it does not age or wear out
– it cannot be deformed or broken
– it cannot be affected by environmental factors
– if deterministic, it always performs
(permanent)

Software may undergo several upgrades during system life cycle
– reliability upgrade – aims to enhance software reliability
  Done by re-designing some modules using better approaches
– feature upgrade – aims to enhance software functionality,
  Likely to increase complexity and thus decrease reliability by
  introducing new bugs

Fixing bugs does not necessarily make software more reliable
– new bugs may be introduced

In 1991, a change of 3 lines of code in a program containing millions lines of code
caused a local telephone system in California to stop

Software is inherently more complex and less regular than hardware
– achieving sufficient test coverage is very difficult

60-65% of software faults originate from
– incomplete, missing, inadequate, inconsistent, unclear requirements

• 35-40% of software faults originate from
  – coding mistakes
  – proportional to
    size of code
    number of paths in code
  Can use software complexity to estimate potential # faults
CSCE 5760: Design For Fault Tolerance

So far we looked at reliability of an individual component or system. In most cases, a system consists of a large number of components. So, we need more complex ways of measuring reliability. Do we need all components to be operational?

Consider a program – it has many execution paths. You may have one path with an fault (not error!) but the fault manifests only once in 1000 executions. We need to include this information in computing reliability of the system.

Consider an example system with N computers, but we only need one of them to be operational for the system to be UP. If we denote the reliability of the system as $R(t)$, then:

$$R(t) = \sum_{i \geq 1} P_i(t)$$

where $P_i(t)$ is the probability that $i$ computers are UP during $[0,t]$.

We can generalize this idea to a measure known as Performability. Let a system can operate at different levels of performance, $L_1, L_2, \ldots, L_k$. For each level $L_j$, we need certain number of computers or performance. Let us define the probability that level $L_j$ fails as $P_{L_j}(t)$. Performability is a vector $[P_{L_1}(t), P_{L_2}(t), \ldots, P_{L_k}(t)]$.

You can think of these levels of performance as different Service Levels in SLA. And may use this to charge a customer based on the levels that were actual delivered.

Note when some of the components fail, we have diminished computational capacity. Let $C_i$ is the computational capacity with $i$ computers. The total capacity of the system during the interval $[0,t]$ is given by:

$$C = \sum_{i \geq 1} C_i P_i(t)$$

We can generalize this idea to a measure known as Performability.

Let a system can operate at different levels of performance, $L_1, L_2, \ldots, L_k$. For each level $L_j$, we need certain number of computers or performance. Let us define the probability that level $L_j$ fails as $P_{L_j}(t)$.

Performability is a vector $[P_{L_1}(t), P_{L_2}(t), \ldots, P_{L_k}(t)]$.

You can think of these levels of performance as different Service Levels in SLA. And may use this to charge a customer based on the levels that were actual delivered.
Let us return to modeling systems with several components.

We can consider an adder with several NAND gates.
We can consider a software system with several modules.

A canonical (or standard) structure is constructed out of N individual modules.
The basic canonical structures are:
- A series system
- A parallel system
- A mixed system

We will assume statistical independence between failures in the individual modules.

Reliability of a Series System

A series system - set of modules so that the failure of any one module causes the entire system to fail.

\[
R_S(t) = \prod_{i=1}^{N} R_i(t)
\]

\(R_i(t)\) is the reliability of module \(i\).

Series System – Modules Have Constant Failure Rates

- Every module \(i\) has a constant failure rate \(\lambda_i\),
  \[
  R_i(t) = e^{-\lambda_i t}
  \]

- \[
  R_S(t) = e^{-\lambda_S t} = e^{-\sum \lambda_i t}
  \]
Series System – Modules Have Constant Failure Rates

Every module $i$ has a constant failure rate $\lambda_i$

$$MTTF_s = \frac{1}{\bar{\lambda}} = \frac{1}{\sum \lambda_i}$$

Note we are adding failure rates (works only for exponential)

Reliability of a Parallel System

- A Parallel System - a set of modules connected so that all the modules must fail before the system fails

$$R_p(t) = 1 - \prod_{i=1}^{N} [1 - R_i(t)]$$

$R_i(t)$ is the reliability of module $i$

Parallel System – Modules have Constant Failure Rates

Module $i$ has a constant failure rate $\lambda_i$

$$R_i(t) = e^{-\lambda_i t} \quad R_p(t) = 1 - \prod_{i=1}^{N} [1 - e^{-\lambda_i t}]$$
Parallel System – Modules have Constant Failure Rates

Example - a parallel system with two modules

\[ R_p(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1 + \lambda_2) t} \]

MTTF of a parallel system with the same \( \lambda \)

\[ MTTF_p = \sum_{i=1}^{N} \frac{1}{i \lambda} \]

Compare this with series system

\[ MTTF_s = \frac{1}{\sum \lambda_i} \]

Non Series/Parallel Systems

Each path represents a configuration allowing the system to operate successfully, e.g., ADF, or ACEF, or BEF

The reliability can be calculated by expanding about a single module \( i \):

\[ R_{\text{system}} = R_i \text{Prob\{System works | i is fault-free\}} \]
\[ + (1-R_i) \text{Prob\{System works | i is faulty\}} \]

Draw two new diagrams: in (a) module \( i \) is operational; in (b) module \( i \) is faulty
Module \( i \) is selected so that the two new diagrams are closer to simple series/parallel structures
CSCE 5760: Design For Fault Tolerance

Consider expanding about C

\[ R_{\text{system}} = R_C \cdot \text{[Reliability when C is working]} + (1 - R_C) \cdot \text{[When C is not working]} \]

Now we need to expand the equation by explore another component, Say E.

\[ R_{\text{system}} = R_C \cdot R_E \cdot \text{[Reliability when E is working]} + (1 - R_E) \cdot \text{[Reliability when E is not working]} + (1 - R_C) \cdot R_E \cdot \text{[Reliability when E is working]} + (1 - R_E) \cdot \text{[Reliability when E is not working]} \]

CSCE5760  September 03, 2019  33

CSCE 5760: Design For Fault Tolerance

Expanding about C and E

\[ R_{\text{system}} = R_C \cdot \text{[Reliability when C is working]} + (1 - R_E) \cdot \text{[Reliability when E is working]} + (1 - R_C) \cdot R_E \cdot \text{[Reliability when E is working]} + (1 - R_E) \cdot \text{[Reliability when E is not working]} \]

CSCE5760  September 03, 2019  34
CSCE 5760: Design For Fault Tolerance

We need to continue this expansion
Until we are only dealing with series or parallel networks

\[ R_{\text{system}} = R_C \left[ R_A R_E (R_B + R_D - R_B R_D) + (1 - R_E) R_A R_F \right] + (1 - R_C) \left[ R_F (R_A R_D + R_B R_E - R_A R_D R_B R_E) \right] \]

If \( R_A = R_B = R_C = R_D = R_E = R_F = R \)
\[ R_{\text{system}} = R^3 (R^3 - 3R^2 + R + 2) \]

If structure is too complicated - derive upper and lower bounds on \( R_{\text{system}} \)
An upper bound - \( R_{\text{system}} \leq 1 - \prod (1 - R_{\text{path}_i}) \)
\( R_{\text{path}_i} \) - reliability of modules in series along path \( i \)
Assuming all paths are in parallel
CSCE 5760: Design For Fault Tolerance

Example - the paths are ADF, BEF and ACE

$R_{\text{system}} \leq 1 - (1 - R_A R_D R_F)(1 - R_B R_E R_F)(1 - R_A R_C R_E R_F)$

If $R_A = R_B = R_C = R_D = R_E = R_F$ then

$$R_{\text{system}} \leq R^3 (R^7 - 2R^4 - R^3 + R + 2)$$

Why is this an upper bound? A component may be in multiple paths. We need to account for double counting its reliability.

How to find a lower bound on reliability

CSCE 5760: Design For Fault Tolerance

A lower bound is calculated based on minimal cut sets of the system diagram. A minimal cut set: a minimal list of modules such that the removal (due to a fault) of all modules will cause a working system to fail.

Minimal cut sets: $F, AB, AE, DE$ and $BCD$

The lower bound is

$$R_{\text{system}} \geq \prod (1 - Q_{\text{cut}_i})$$

$Q_{\text{cut}_i}$ - probability that the minimal cut $i$ is faulty (i.e., all its modules are faulty)

Example - $R_A = R_B = R_C = R_D = R_E = R_F$

$$R_{\text{system}} \geq R^5 (24 - R^5 + 9R^4 - 33R^3 + 62R^2 - 60R)$$
CSCE 5760: Design For Fault Tolerance

Example – Comparison of Bounds
Example - -  \( R_A=R_B=R_C=R_D=R_E=R \)
Lower bound here is a very good estimate for a high-reliability system
Conservative and a safer estimate

Exercise 2.14

Let us consider the following system
D is bidirectional.
Let us write \( R_{\text{system}} \) expression

Condition on the state of module D. If D is down, we reason as follows:
The system consists of two parallel connections;
\{A; B; E\} and \{C; F\}
\{C; F\} will be connected only if both are up, so the reliability of this is \( R_{CF}(t) = R_C(t)R_F(t) \)
CSCE 5760: Design For Fault Tolerance

Exercise 2.14

For \{A, B, E\}:
We need either \{A\}, \{B\} or both PLUS \{E\} working

\[ R_{A} = 1 - (1 - R_{A}(t))(1 - R_{B}(t)) \]
\[ R_{AB} = R_{A} R_{B} \]
\[ R_{D, down}(t) = 1 - (1 - R_{ABE}(t))(1 - R_{D}(t)) \]

Now consider when D is up
We can view our system as: \{A, B, C\} in parallel connected to \{E, F\} in parallel by D
So, \[ R_{D, up} = R_{ABC}(t) R_{EF}(t) \]

\[ R_{A}(t) = 1 - (1 - R_{A}(t)) \]
\[ R_{B}(t) = 1 - (1 - R_{B}(t)) \]
\[ R_{C}(t) = 1 - (1 - R_{C}(t)) \]
\[ R_{D}(t) = 1 - (1 - R_{D}(t)) \]
\[ R_{system}(t) = R_{D, down}(t) R_{EF}(t) + R_{D, up}(t) R_{D}(t) \]

CSCE 5760: Design For Fault Tolerance

Redundancy And Resiliency: Generic M-of-N Systems

An M-of-N system consists of N identical modules

Fails when fewer than M modules are functional
Best-known example - The Triplex (TMR)
Three identical modules whose outputs are voted on
This is a 2-of-3 system: as long as a majority of the processors produce correct results, the system will be functional

Reliability of M-of-N Systems

\[ N \] identical modules
\[ R(t) \] - reliability of an individual module

The reliability of the system is the probability that \[ N-M \] or fewer modules have failed by time \( t \) (or – at least M are functional)

\[ R_{M-of-N}(t) = \sum_{i=0}^{N-M} \binom{N}{i} (1 - R(t))^i R(t)^{N-i} \]
Reliability of M-of-N Systems

\[ R_{m-of-N}(t) = \sum_{i=0}^{N-M} \binom{N}{i} (1 - R(t))^i R(t)^{N-i} \]

\[ = \sum_{i=M}^{N} \binom{N}{i} R(t)^i (1 - R(t))^{N-i} \]

Where \( \binom{N}{i} = \frac{N!}{i!(N-i)!} \)

Can we assume that all modules behave independently?
Is it possible to have correlated failures?

The same design failure may exist in several replicated components?

Correlated failure can greatly diminish reliability

Example: Let \( q_{cor} \) be the probability that the entire system suffers a global failure

\[ R_{m-of-N-corr}(t) = (1 - q_{cor}) \sum_{i=M}^{N} \binom{N}{i} R(t)^i (1 - R(t))^{N-i} \]

- If system is not designed carefully, the correlated failure factor can dominate the overall failure probability
- Different modes of correlation among modules exist - not necessarily a global failure
- Correlated failure rates are extremely difficult to estimate
- From now on we will assume statistically independent failures in modules
A common NMR (M out of N) system is the Triple Modular redundancy (TMR)
Here N=3 and M = 2 (2 out of 3)
A voter picks the majority output
Voter can fail - reliability of voter $R_{vot}(t)$

Reliability of a TMR

$$R_{tmr}(t) = R_{vot}(t) \sum_{i=0}^{3} \binom{3}{i} (1-R(t))^i R(t)^{3-i}$$

$$= R_{vot}(t) \sum_{i=2}^{3} \binom{3}{i} R(t)^i (1-R(t))^{3-i}$$

$$= R_{vot}(t) \left( 3R^2(t) - 2R^3(t) \right)$$

We can derive this relationship using other stochastic models
In particular, Markov systems

Reliability of TMR - Constant Failure Rates \[^{\rightarrow}\] as a probability distribution

$$R(t) = e^{-\lambda t}$$

- Assuming no voter failures - $R_{vot}(t)=1$

$$R_{tmr}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

$$MTTF_{tmr} = \int_0^\infty R_{tmr}(t) \cdot dt = \frac{5}{6\lambda} \leq \frac{1}{\lambda} = MTTF_{simplex}$$

Not unexpected, but now the failure rate is lower than the failure rate of a single unit.

However, if the individual component reliability is low, TMR may actually be a disadvantage
CSCE 5760: Design For Fault Tolerance

- M-of-N cluster with N odd and $M = (N+1)/2$
- Assume voter failure rate negligible - $R_{vot}(t)=1$

- Below $R=0.5$ - redundancy becomes a disadvantage
- Usually $R >> 0.5$ - triplex offers significant reliability gains