CSCE 5760: Design For Fault Tolerance

Solutions to Midterm

1. (30%) In trying to assess Bernie Sanders chances for nomination by the Democratic party, Sanders's supporters used a Markov process. His chances are based on wins and losses at various primaries. Due to the media coverage, the win or loss at one primary will influence Sanders's winning chances at the next primary. The following Markov states are used: sure Winner (W), Improving chances (I), Sinking (S) and sure Loser (L). The following are the state transition probabilities: W-W = 1; L-L = 1; I-W = 0.3, I-I = 0.3, I-S = 0.3, I-L = 0.1; S-I = 0.2, S-S = 0.5 and S-L = 0.3.

a. Draw a state diagram.

b. Classify the states.

W and L are absorbing states while I and S are transient states.

c. Assuming that Bernie Sanders starts with equal probability in a transient state, what are the probabilities of winning and losing the nomination?

We need to construct the Fundamental Matrix from (I-Q)^{-1}

Then compute F = M*R

Assuming we have 50% probability of starting either in I or S, then the probability of Winning

= 0.5*0.516+0.5*0.207 = 0.3615

Likewise, the probability of Losing

= 0.5*0.481+0.5*0.798 = 0.6395

Note these probabilities must add up to 1. (These are made up numbers)
2. Write the expression for reliability for the following system

We need to reduce the 3 R3 in parallel first.
Then we combine this with R2 in series. We will have R4 and R5 in series
We reduce the two parallel paths and combine with R1
If I did not make any errors in manipulation we get
\[ R_1 - R_1 R_2 + R_1 R_2 (R_3)^3 + 3R_1 R_2 R_3 \]
\[ 3 R_1 R_2 R_3^2 + 3R_1 R_2 R_4 R_5 (R_3)^3 + 3R_1 R_2 R_4 R_5 (R_3)^2 \]
If we set all R to equal, we get
\[ R = R^3 + 2R^3 + 2R^3 + 3R^3 - R \]

3. Let C be a linear code over GF(3) – that is each dimension can be (0,1,2) and the following is the generator matrix for the code

\[
\begin{bmatrix}
1 & 1 & 1 & 0 \\
2 & 0 & 1 & 1 \\
\end{bmatrix}
\]

a). How many data bits and parity bits are in this code?
I should have asked for how many digits (not bits)
Since is a 2x4 Generator matrix, we have 2 data digits and 2 parity digits

b). How many code words are in this system?
Since there are 2 code words we 3^2 or 9 code words

b). Find the Parity Check matrix for this generator matrix.
Remember to convert the G matrix into standard matrix we need [I|A]

\[
\begin{bmatrix}
1 & 0 & 1 & 1 \\
0 & 1 & 2 & 1 \\
\end{bmatrix}
\]
CSCE 5760: Design For Fault Tolerance

Note: We have used used $H = [A^T | I]$ we get

\[
\begin{array}{ccc}
1 & 2 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

This is OK in binary. But to be more accurate we need $[-A^T | I]$. In non-binary systems the negative sign is important. Now we have $H$ as

\[
\begin{array}{ccc}
-1 & -2 & 1 \\
-1 & -1 & 0 \\
\end{array}
\]

If we remember the addition (and multiplication tables for $\text{GF}_3$) then $-1 = 2$ and $-2 = 1$

\[
\begin{array}{ccc}
2 & 1 & 1 \\
2 & 2 & 0 \\
\end{array}
\]

4. Given $X^7 - 1 = (X+1)(X^3+X+1)(X^3+X^2+1)$ and assume that we are using

For each of the factors of $X^7 - 1$, identify the $(n,k)$ codes that can be generated. That is how many data bits, how many parity bits and how many bit errors that can be corrected

$X+1$ generates $(7,6)$ code; remember this means, we have 6 data bits and this code can only detect one bit error. Since we have 6 data bits, we have $2^6 = 64$ codewords

$(X^3+X+1)$ defines $(7,4)$ code and this can detect 2 consecutive bit errors. Since we have 4 data bits, this generates 16 codewords.

$(X^3+X^2+1)$ also defines a $(7,4)$ code which can detect 2 consecutive bit errors. This also generates 16 codewords (4 data bits).

Likewise we can also use $(X+1)(X^3+X+1) = X^4+X^3+X^2+1$ which generates $(7,3)$ code with 3 data bits (detect 3 consecutive bit errors) and has 8 codewords

$(X+1)^*(X^3+X^2+1)=X^6+X^5+X^4+X^3+X+1$ also generates $(7,3)$ code and has 8 codewords

Finally $(X^3+X^2+1)^*(X^3+X^2+1)^* = X^6+X^5+X^4+X^3+X^2+X+1$ generates $(7,1)$ code - only one data bit and thus generates only 2 codewords
CSCE 5760: Design For Fault Tolerance

Review

RAID 1-10

Disk 0 Disk 1 Disk 2 Disk 3

Disk 0 Disk 1 Disk 2 Disk 3 Disk 4 Disk 5 Disk 6

RAID 3

RAID 4

Disk 0 Disk 1 Disk 2 Disk 3

Disk 0 Disk 1 Disk 2 Disk 3

CSCE 5760: Design For Fault Tolerance

RAID 5

Disk 0 Disk 1 Disk 2 Disk 3

Disk 0 Disk 1 Disk 2 Disk 3 Disk 4

RAID 6

RAID 1

RAID 1

CSCE 5760: Design For Fault Tolerance

CSCE 5760: Design For Fault Tolerance
CSCE 5760: Design For Fault Tolerance

RAID Issues

Rebuild times can be long
- Cause stress on other drives causing secondary failure
- Can disrupt normal system performance due to extra disk activity during rebuild

Secondary failure in RAID 3, 4, and 5 causes complete failure of the array and total data loss

Drives are generally same age in a RAID
- Probability of 2 drives in a RAID failing within 1 hour of each other is 4 times higher than independent systems. (Thus the development of RAID 6)
- Have to use correlated failures analysis

Reliability evaluation of RAIDs

Consider RAID 1 (mirrored disks)
This can be modeled using Markov states
2 working, 1 working zero working
We permit repair

\[
\frac{dP_2(t)}{dt} = -2\lambda P_2(t) + \mu P_1(t) \\
\frac{dP_1(t)}{dt} = -(\lambda + \mu)P_1(t) + 2\lambda P_2(t) \\
P_0(t) = 1 - P_1(t) - P_2(t) \\
P_2(0) = 1; \quad P_0(0) = P_1(0) = 0
\]

\[
R(t) = P_1(t) + P_2(t) = 1 - P_0(t)
\]

How to find MTTF?

Textbook uses a slightly different way to obtain MTTF
calls it Mean Time To Data Loss (MTTDL)
We will assume that repair rate (\(\mu\)) is much larger (faster) than failure rate (\(\lambda\))
CSCE 5760: Design For Fault Tolerance

Starting in state 2 at t=0
So we need to compute: Time to go from 2 to 1 and then 1 to 0
- mean time before entering state 1 = 1/(2\(\lambda\))
  
  Note 2\(\lambda\) is the rate of leaving state 2
Likewise mean time spent in state 1 is 1/(\(\lambda + \mu\)) before leaving state 1
But from state 1, we may return to state 2
Go back to state 2 with probability q = \(\mu / (\mu + \lambda)\)
or to state 0 with probability p = \(\lambda / (\mu + \lambda)\)
Probability of n visits to state 1 before transition to state 0 is \(q^{n-1}p\)
Mean time to enter state 0:
\[
T_{2\rightarrow 0}(n) = n \left( \frac{1}{2\lambda} + \frac{1}{\lambda + \mu} \right) = n \frac{3\lambda + \mu}{2\lambda(\lambda + \mu)}
\]
This gives the time when we visit 2 n times

\[
MTTDL = \sum_{n=1}^{\infty} q^{n-1} p T_{2\rightarrow 0}(n) = \sum_{n=1}^{\infty} q^n p T_{2\rightarrow 0}(1) = \frac{T_{2\rightarrow 0}(1)}{p} = \frac{3\lambda + \mu}{2\lambda^2}
\]

CSCE5760  October 22, 2019  11

CSCE 5760: Design For Fault Tolerance

If \(\mu >> \lambda\), the transition rate into state 0 from either states 1 or 2 is approximately the 1/MTTDL
(which can be approximated as MTTF)
Reliability of RAID 1
\[
R(t) = e^{-t/MTTDL}
\]

Impact of Disk failure rates

Impact of Disk Repair rates
Availability Calculation.
Note we will construct a Markov chain with no absorbing states

\[ P_2(t) = \frac{\mu^2}{(\lambda + \mu)^2} + 2\lambda\mu/(\lambda + \mu)^2 e^{-(\lambda+\mu)t} + \lambda^2/(\lambda + \mu)^2 e^{-2(\lambda+\mu)t} \]

\[ P_1(t) = 2\lambda\mu/(\lambda + \mu)^2 + 2\lambda(\lambda - \mu)/(\lambda + \mu)^2 e^{-(\lambda+\mu)t} - 2\lambda^2/(\lambda + \mu)^2 e^{-2(\lambda+\mu)t} \]

\[ P_0(t) = 1 - P_2(t) - P_1(t) \]

Long term availability = 1 - P_0 = P_2 + P_1

\[ = (\mu^2 + 2\lambda\mu)/(\lambda + \mu)^2 = 1 - \lambda^2/(\lambda + \mu)^2 \]

Likewise we can model other RAID systems

---

CSCE 5760: Design For Fault Tolerance

Should you replicate data or use error correcting codes?

Let us consider distributed file systems – files are distributed on network such as Hadoop file system

Let us consider replication – We make a copy of the file at each node in the system

Reads can be satisfied by any copy

But on write, we need to update every copy

Example - five copies in five nodes:

If A is disconnected and a write updates the copy in A - the rest no longer consistent with A

Any read of their data will result in stale data

How many copies should we read (write)?
CSCE 5760: Design For Fault Tolerance

Assign \( v \) votes to copy \( i \) of the data

- \( S \) - set of all nodes with copies of the data
- \( V \) - sum of all votes

\[ V = \sum_{i \in S} V_i \]

\( r, w \) (refer to how many reads and writes are needed) - variables such that \( r + w > V \); \( w > V/2 \)

\[ V(X) - \text{total number of votes assigned to copies in set } X \quad \Rightarrow \quad V(X) = \sum_{i \in X} V_i \]

Strategy ensuring that all reads use the latest data

To complete a read - read nodes of a set \( R \subseteq S \) such that \( V(R) \geq r \)

To complete a write - write on every node of a set \( W \subseteq S \) such that \( V(W) \geq w \)

Since \( r+w > V \) and \( w > V/2 \), we can show that writing to sufficient copies with total votes of \( w \), will assure that reading copies with total votes of \( r \) is satisfied.

Example, let us assume each node has one vote

Sum of votes for a datum = 5

- \( w > 5/2 \) and \( r > 5-w \)

So, possible values for \( r \) and \( w \) are

- \((1,5), (2,5), (3,5), (4,5), (5,5)\)
- \((2,4), (3,4), (4,4), (5,4), (3,3)\)

Consider \((r,w) = (1,5)\). Read can be satisfied by any copy. But we must update all copies

Read is fast but write is slow. Also if one node say A fails, we cannot satisfy writes since we cannot update all copies

Consider \((r,w) = (3,3)\). Now read is slow (need to read 3 copies and vote)

Write is a bit faster \(\Rightarrow\) need to update only 3 copies

If one node fails, we can still meet \((r,w) = (3,3)\)
How to compute Reliability and Availability?

We can use Markov models
For example if (r,w) = (3,3) and we have 5 nodes

\[ \text{Reliability} = 1 - P_F(t) \]

Another issue to worry about is: if different nodes and different links have different reliabilities (or failure rates).

We want to maximize “point” or instantaneous availability. Assign higher weights (or votes) to more reliable nodes

Then we define a quorum → minimum votes for either read or write

Optimal assignment of votes is very complex
So the book suggests two different heuristics

If we assume \( a_n(i) \) is the availability of node \( i \)
And \( a_l(j) \) is the availability of link \( j \).

Heuristic -1 → votes assigned node \( i \) = node-\( i \) availability \* (sum of link availabilities)
rounded to nearest integer

\[ V(i) = a_n(i) \sum_{j \in L(i)} a_l(j) \]

\( L(i) \) is the set of links connecting to node \( i \)
If all \( V(i) \)s are even, increase the largest \( V(i) \) by 1

Heuristic -2. Let \( k(i,j) \) is the node connected to node \( i \) via link \( j \)
Here we give votes to node \( i \) based on its impact on other nodes and links

\[ V(i) = a_n(i) + \sum_{j \in L(i)} a_l(j) \cdot a_n(k(i,j)) \]
CSCE 5760: Design or Fault Tolerance

Example. Consider the following configuration

Using Heuristic 1

\[ v(A) = \text{round}(0.7 \times 0.7) = 0 \]
\[ v(B) = \text{round}(0.8 \times 1.8) = 1 \]
\[ v(C) = \text{round}(0.9 \times 1.6) = 1 \]
\[ v(D) = \text{round}(0.7 \times 0.9) = 1 \]

Sum of votes \( V = 3 \)

So, \( w > V/2 \), \( w \) must be either 2 or 3

If \( w=2 \), \( r \) must be 2; \( \text{(note } w > V/2 \text{ and } r > V-w) \)

now the read quorum can be BC, CD or BD

same for write quorum

if \( w=3 \), \( r \) can be either 1 or 2 \( \text{(note } w > V/2 \text{ and } r > V-w) \)

write quorum is BCD

if \( r=1 \), then we can define B, C or D as read quorum

CSCE 5760: Design or Fault Tolerance

Heuristic 2

\[ v(A) = \text{round}(0.7 + 0.7 \times 0.8) = 1 \]
\[ v(B) = \text{round}(0.8 + 0.7 \times 0.7 + 0.9 \times 0.9 + 0.2 \times 0.7) = 2 \]
\[ v(C) = \text{round}(0.9 + 0.9 \times 0.8 + 0.7 \times 0.7) = 2 \]
\[ v(D) = \text{round}(0.7 + 0.2 \times 0.8 + 0.7 \times 0.9) = 1 \]

Here the total votes is 6 or even. We increase the votes for one of the larger nodes, say \( V(b) = 3 \)

\( V=7 \), so \( w > 7/2 \), either 4, 5, 6 or 7

We can first decide on \( w \) and then assign values for \( r \)

If \( w = 4 \), \( r \) can be 4 (and no other value) \( \text{(note } w > V/2 \text{ and } r > V-w) \)

If \( w = 5 \), \( r \) can be 2, 3, 4 or ..., etc

For each \( r \) and \( w \) values, we can define read and write quorums
CSCE 5760: Design or Fault Tolerance

The following are some possible quorum assignments for $r+w=8$
Remember $V(A)=1; V(B)=3, V(C)=2, V(D)=1$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$w$</th>
<th>Read Quorums</th>
<th>Write Quorums</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>AB, BC, BD, ACD</td>
<td>AB, BC, BD, ACD</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>B, AC, CD</td>
<td>BC, ABD</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>B, C, AD</td>
<td>ABC, BCD</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>A, B, C, D</td>
<td>ABCD</td>
</tr>
</tbody>
</table>

Once we defined quorums, we can calculate availability
For example say our read quorum is (AB, BC, BD, ACD).

The system is available as long as one of the quorums is available.

CSCE 5760: Design or Fault Tolerance

One final topic from Chapter 3
Algorithm based fault tolerance
This applies not only to replicas but a general fault tolerance mechanism

You include some redundancy in your data structures
For example, with any given structure, you define a “check sum” as an added element

Consider Matrices –
we can add a column that is a checksum on row elements
we can add an extra row which is a checksum on column elements

This way, when we have matrix algorithms (such as matrix multiplication), we can use these checksums to correct errors.

$$A_F = \begin{bmatrix} A & Af \\ eA & eAf \end{bmatrix}$$

- $A_C + B_C = C_C$
- $A_R + B_R = C_R$
- $A_F + B_F = C_F$
- $A_B R = C_R$
- $A_C B = C_C$
- $A_C B_R = C_F$

$eA$ is the checksum on columns; $Af$ is the checksum on rows
CSCE 5760: Design or Fault Tolerance

Read Chapter 3 for more examples of Algorithm Based Fault Tolerance

One of my interest is to explore (or categorize) fault tolerance techniques
Explore which is suitable under what conditions
particularly based on user specified levels of reliability

Chapter 4 – Network reliability
I will skip some parts of the chapter and return to them if we have time

Network may be between cores on a multicore, between cores and memory
between nodes in a cluster, or other

Components: nodes, links, switches, routers → any of them may fail

Some measures based on viewing networks as graphs
we can have either uni-directional or bi-directional links

Path – how many paths exist between two nodes

Node (link) connectivity – the minimum number of nodes (links) when removed
disconnects the network

Distance – minimum distance between a pair of nodes
Diameter – maximum of the distances

Diameter stability – when links and nodes fail, the diameter may increase

What is the minimum number of nodes that must fail before the diameter increases
and what is the rate of diameter change

We can define a probability distribution (discrete)
\( (P_{d+1}, P_{d+2}, \ldots, P_{\text{infinity}}) \) → probability of increasing diameter by 1, 2, 3…
Textbook spends a lot of time evaluating the reliability of multistage networks like butterfly (or shuffle-exchange), hypercube etc.

A quick introduction of multistage with butterfly network
I will leave it up to you to read other networks.

We start with a simple 2x2 switch
The switch can be in one of 4 possible modes:

We can use these switches to build network for connecting nodes
Consider a k-stage network

We connect $2^k$ nodes (say processors) to $2^k$ nodes (processors or memory modules)
One interesting observation is that a k-stage butterfly network contains $2^{k-1}$ stage networks.
Failure of a switch will disconnect the network!
One way to improve the reliability is to add an extra stage. See for example

Now we can reroute around the failed switch

Measuring the effectiveness of these networks.
The book first calculates the bandwidth supported by failure-free network

Then calculate the bandwidth in the presence of failures

Let us assume each processor generates a request (textbook looks at Memory requests) each cycle with a probability of $p_r$. If we have $N$ units (say memory modules), the probability of generating a request to each unit is $p_r/N$

Assuming independence of requests and all processors are identical for butterfly network (because of the recursive nature)

Let $p_{ri}$ is the probability that an output of a switch in stage $i$ carries a request note $i$ ranges from 0, 1, ..., $k-1$ ($k-1$ is the first stage)

Starting at first stage (or $k-1$), there are two requests for each switch box from the two processors connected
CSCE 5760: Design or Fault Tolerance

Each processor generates request equally likely to all processors
or each processor’s request goes to an output line of the box = p_r/2
Since we have two processors p_r^(k-1) = p_r/2 + p_r/2 - (p_r/2)^2 = p_r - (p_r^2)/4
Now we can develop a recursive formula

P_r(i-1) = p_r(i) - (p_r(i)^2)/4

Note that the final switch box of interest is in stage zero.
So we need to find p_r(0)
Since we have N processors generating request the final bandwidth supported = N*p_r(0)
If we want to generate formulas for the bandwidth in presence of failures
we need to include that the request at any stage i is processed
when the switch in that stage fails.

or p_r(i) will become p_f * p_r(i) where p_f is the probability of a switch working
Likewise we can include link failures

Another measure of network effectiveness is connectivity
In other words, compute how many connections can be maintained in the presence of failures

In a 2^k * 2^k butterfly network each connection (from a processor to a memory unit)
must traverse k+1 links and k switches
If we assume that the probability a switch or a link has not failed as p_s and p_l

What is the probability that a processor can successfully connect to a memory unit?
p_l*(k+1) * p_s

Since there are 2^k processors and 2^k memory unit, the number processor-memory pairs connected is given by Q = 2^k * p_l^(k+1) * p_s

This provides a measure on the overall network performance
We can also find how many processors are actually able to connect to memories
CSCE 5760: Design or Fault Tolerance

For this computation, the book only assumes links fail and switches do not fail.

A processor is available if it can connect to at least a memory unit. Likewise a memory is available if it can be connected by at least one processor.

Let us start in zero stage (output stage) where the outputs of switches connect to memory.

For a processor to connect to at least one memory unit, at least one output of the switch in the last stage must be working (that is $q_l = 1 - p_l$).

$$\phi(0) = (1 - q_l^2)$$

Now we have to compute this value for first stage (k-1 stage) recursively.

For output in ith stage to work, the inputs to that stage should be working.

A switch box in ith stage connects to two inputs in i-1 stage.

The book also shows how to compute these probabilities when we have extra stages.
CSCE 5760: Design or Fault Tolerance

Consider 2-D grid networks

We have some alternate paths
But switch failures disconnect some processor pairs
Can we consider building redundancy – somewhat like extra stage network?

We added an extra row and extra column of switches
If a switch in a row or a column fails, the spare switch in that row or column is used to route messages
CSCE 5760: Design or Fault Tolerance

Here we use a spare that can be used to replace one of 4 switches.

This is better because the spare is closer – as compared to column or row spares.

CSCE 5760: Design or Fault Tolerance

In this case, each node can use up to 4 spares – but higher redundancy.

If you are interested, read textbook to compute reliabilities of mesh networks.

What about hypercube?
Can we add redundancy?

See page 129.
Cube connected networks – each node is replaced by multiple nodes.

In general we can have $k$ nodes replacing each node a hypercube.

Total nodes will be $K \times 2^d$.

We can used these types of nodes to reduce cost of the network or to improve reliability.

These are called Chordal rings.

Note that we can use these additional links to minimize diameter.
CSCE 5760: Design or Fault Tolerance

Finding path (or end-to-end) reliability in a network
Need to find all possible paths

Reliability between \( N_1 \) to \( N_4 \)
Three paths
\[
P_1 = \{X_{1.2}, X_{2.4}\}
\]
\[
P_2 = \{X_{1.3}, X_{3.4}\}
\]
\[
P_3 = \{X_{1.2}, X_{2.3}, X_{3.4}\}
\]

We cannot simply add the reliabilities of these paths, since there are some common nodes and links and the reliabilities of those will be counted more than once

We can use the idea of representing states for all possible failures of links or nodes
In this example we can also explore mutually exclusive events

Path 1 is operational
Path 1 is not operational but path 2 is operational
Paths 1 and 2 are not operational but path 3 is operational

\[
R_{N_1,N_4} = p_{1.2} p_{2.4} + p_{1.3} p_{3.4} [1 - p_{1.2} p_{2.4}] + p_{1.2} p_{2.3} p_{3.4} [q_{1.3} q_{2.4}]
\]

\[
q_{1,3} = (1-p_{1.3})
\]

\[
E_1 \cup E_2 \cup \cdots \cup E_m = E_1 \cup (E_2 \cap \bar{E}_1) \cup (E_3 \cap E_1 \cap \bar{E}_2) \cup \cdots \cup (E_m \cap E_1 \cap E_2 \cap \cdots \cap \bar{E}_{m-1})
\]

\[
R_{N_1,N_4} = \text{Prob}[E_1] + \text{Prob}[E_2 \cap \bar{E}_1] + \cdots + \text{Prob}[E_m \cap E_1 \cap E_2 \cap \cdots \cap \bar{E}_{m-1}]
\]
CSCE 5760: Design or Fault Tolerance

Consider problem 4.15 from book

All the links in a given 3 x 3 torus network are directed as shown in the diagram below. Calculate the path reliability for the source node 1 and the destination node 0. Denote by $p_{ij}$ the probability that the link from node $i$ to node $j$ is operational and assume that all nodes are fault-free.

Need to find how many paths are available between 1 to 0. If all paths fail, the connection fails

There are 4 paths –note links are NOT bi-directional

$P_1 = \{0-1\}$
$P_2 = \{1-7 7-4 4-3 3-0\}$

$P_3 = \{1-7 7-6 6-3 3-0\}$
$P_4 = \{1-7 7-6 6-8 8-5 5-4 4-3 3-0\}$

Again we can either states to represent link failures

Or define the 4 paths as 4 events and define mutually exclusive events

$P_1$ is operational
$P_2$ is operational but $P_1$ is not
$P_3$ is operational but both $P_1$ and $P_2$ are not
$P_4$ is operational but $P_1$, $P_2$, and $P_3$ are not

In general, we need to be able detect failures on links (or switches)
and route messages/request through alternate routes

Timeout can be used to detect failure of a path
Chapter 5 Software Fault tolerance

Why do software contain bugs
Design failures
Error in coding

Note some consider not meeting requirements as failure – we will not

Testing – can we assure that testing leads to fault free software?

Path testing – test for all possible control flow paths
Integration testing – test all modules

With more complex software, exhaustive testing impractical
Even generating test patterns is complex

Can we mathematically prove a program (at least as designed) is correct?
Again can be very complex

Modeling software using formal logic (Model Checking)
predicate logic and higher logics (and theorem proving tools)
or petri nets
of dataflow graphs

If we have time, I will return to these formal models

So we have to see how we can make software fault tolerant?

First we need to be able detect failures
incorrect results?
out of range values?

Acceptance tests
How do we know what are acceptable results?
Can use models or “common sense”
-40 C in summer may mean that something is wrong
CSCE 5760: Design or Fault Tolerance

Timing Checks. If we have a good idea on how long a software system takes, we can use timers to detect errors such as infinite loops or some other reasons for extended execution times.

Random or probabilistic checks

Textbook gives an example

Consider Matrix multiplication: A*B = C

Consider a vector V generated randomly.

Then compute

A*(B*V) and C*V

If these resulting vectors are equal, you have high confidence that C is the correct result

But we are doing additional computations

Original computation = O(n^3)
Extra computation = O(n^2)

CSCE 5760: Design or Fault Tolerance

Verification of Inputs and Outputs

Make sure inputs are within acceptable values
Check for the acceptability of outputs

Inputs – domain of values
Outputs – Range of values

Can include checks of range of values with all program variables
bounds on temperature
bounds on vehicle speeds

Two issues to keep in mind with bounds

Sensitivity: Conditional probability that bounds check fails given that the output is in error – or detect all errors

Specificity: conditional probability that output is in error, given that the bounds check failed: Do not generate false alarms – or reject correct output
CSCE 5760: Design or Fault Tolerance

How to implement acceptance tests?

Use wrappers

The wrapper filters inputs and outputs

Textbook gives a few examples

Buffer Overflow, EDF scheduler

C runtime system does not very address ranges.

you can allocate an array with n elements but access beyond n

Likewise, no checks are made to see if you are accessing beyond current
top of the stack

Stack is used for storing return addresses

Wrapper can be used to verify that accesses are limited to currently allocated memory
areas only

Other examples:

Known limitations of software
does not work for some inputs
wrapper checks for the validity of inputs

Factors influencing capabilities of acceptance tests

Acceptance tests are mostly application dependent

If the application is available only as a black box (no source code or internal
details), acceptance tests will be limited

If the wrapped software is well tested, may have information about the software
behavior, previous detected errors etc.

Software Rejuvenation

Periodically “update” software
Examples given in book are not great
reboot, reclaim garbage, kill zombie tasks