

FAULT TOLERANT SYSTEMS

<http://www.ecs.umass.edu/ece/koren/FaultTolerantSystems>

Part 1 - Introduction
Chapter 1 - Preliminaries

Part.1 .1

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Prerequisites

- ◆ **Basic courses in**
 - * **Digital Design**
 - * **Hardware Organization/Computer Architecture**
 - * **Probability**

Part.1 .2

Copyright 2007 Koren & Krishna, Morgan-Kaufman

References

◆ Main reference

- * I. Koren and C. M. Krishna, *Fault Tolerant Systems*, Morgan-Kaufman 2007.

◆ Further Readings

- * M.L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*, Wiley, 2002, ISBN 0-471-29342-3.
- * D.P. Siewiorek and R.S. Swarz, *Reliable Computer Systems: Design and Evaluation*, A.K. Peters, 1998.
- * D.K. Pradhan (ed.), *Fault Tolerant Computer System Design*, Prentice-Hall, 1996.
- * B.W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.

Administrative Details

- ◆ **Instructor:** Prof. Israel Koren
- ◆ **Office:** KEB 309E, Tel. 545-2643
- ◆ **Email:** koren@ecs.umass.edu
- ◆ **Office Hours:** TuTh 4:00-5:00pm
- ◆ **Course web page:**
<http://euler.ecs.umass.edu/ece655/>
- ◆ **Grading:**
 - Homework - 10%
 - Mid-term I - 25%
 - Mid-term II - 25%
 - Project or Seminar - 40%

Course Outline

- ◆ Introduction - Basic concepts
- ◆ Dependability measures
- ◆ Redundancy techniques
- ◆ Hardware fault tolerance
- ◆ Error detecting and correcting codes
- ◆ Redundant disks (RAID)
- ◆ Fault-tolerant networks
- ◆ Software fault tolerance
- ◆ Checkpointing
- ◆ Case studies of fault-tolerant systems
- ◆ Defect tolerance in VLSI circuits
- ◆ Fault detection in cryptographic systems

Part.1 .5

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Fault Tolerance - Basic definition

- ◆ Fault-tolerant systems - **ideally** systems capable of executing their tasks correctly regardless of either hardware failures or software errors
- ◆ **In practice** - we can never guarantee the flawless execution of tasks under any circumstances
- ◆ Limit ourselves to types of failures and errors which are more likely to occur

Part.1 .6

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Need For Fault Tolerance

- ◆ 1.
- ◆ 2.
- ◆ 3.

Need For Fault Tolerance - Critical Applications

Aircrafts, nuclear reactors, chemical plants, medical equipment

- ◆ **A malfunction of a computer in such applications can lead to catastrophe**
- ◆ **Their probability of failure must be extremely low, possibly one in a billion per hour of operation**
- ◆ **Also included - financial applications**

Need for Fault Tolerance - Harsh Environments

- ◆ **A computing system operating in a harsh environment where it is subjected to**
 - * electromagnetic disturbances
 - * particle hits and alike
- ◆ **Very large number of failures means: the system will not produce useful results unless some fault-tolerance is incorporated**

Need For Fault Tolerance - Highly Complex Systems

- ◆ **Complex systems consist of millions of devices**
- ◆ **Every physical device has a certain probability of failure**
- ◆ **A very large number of devices implies that the likelihood of failures is high**
- ◆ **The system will experience faults at such a frequency which renders it useless**

Hardware Faults Classification

- ◆ Three types of faults:
- ◆ **Transient Faults** - disappear after a relatively short time
 - * **Example** - a memory cell whose contents are changed spuriously due to some electromagnetic interference
 - * Overwriting the memory cell with the right content will make the fault go away
- ◆ **Permanent Faults** - never go away, component has to be repaired or replaced
- ◆ **Intermittent Faults** - cycle between active and benign states
 - * **Example** - a loose connection
- ◆ Another classification: Benign vs malicious

Faults Vs. Errors

- ◆ **Fault** - either a hardware defect or a software/programming mistake
- ◆ **Error** - a manifestation of a fault
- ◆ **Example:** An adder circuit with one output lines stuck at 1
- ◆ This is a fault, but not (yet) an error
- ◆ Becomes an error when the adder is used and the result on that line should be 0

Propagation of Faults and Errors

- ◆ Faults and errors can spread throughout the system
 - * If a chip shorts out power to ground, it may cause nearby chips to fail as well
- ◆ Errors can spread - output of one unit is frequently used as input by other units
 - * Adder example: erroneous result of faulty adder can be fed into further calculations, thus propagating the error

Redundancy

- ◆ **Redundancy** is at the heart of fault tolerance
- ◆ Redundancy - incorporation of extra components in the design of a system so that its function is not impaired in the event of a failure
- ◆ We will study four forms of redundancy:
 - * 1.
 - * 2.
 - * 3.
 - * 4.

Hardware Redundancy

- ◆ Extra hardware is added to override the effects of a failed component
- ◆ **Static Hardware Redundancy** - for immediate masking of a failure
 - * **Example:** Use three processors and vote on the result. The wrong output of a single faulty processor is masked
- ◆ **Dynamic Hardware Redundancy** - Spare components are activated upon the failure of a currently active component
- ◆ **Hybrid Hardware Redundancy** - A combination of static and dynamic redundancy techniques

Software Redundancy - Example

- ◆ Multiple teams of programmers
- ◆ Write different versions of software for the same function
- ◆ The hope is that such diversity will ensure that not all the copies will fail on the same set of input data

Information Redundancy

- ◆ Add check bits to original data bits so that an error in the data bits can be detected and even corrected
- ◆ Error detecting and correcting codes have been developed and are being used
- ◆ Information redundancy often requires hardware redundancy to process the additional check bits

Time Redundancy

- ◆ Provide additional time during which a failed execution can be repeated
- ◆ Most failures are transient - they go away after some time
- ◆ If enough slack time is available, failed unit can recover and redo affected computation

Fault Tolerance Measures

- ◆ It is important to have proper yardsticks - **measures** - by which to measure the effect of fault tolerance
- ◆ A **measure** is a mathematical abstraction, which expresses only some subset of the object's nature
- ◆ Measures?
 - *
 - *
 - *

Traditional Measures - Reliability

- ◆ **Assumption:**
The system can be in one of two states:
"up" or "down"
- ◆ **Examples:**
 - * Lightbulb - good or burned out
 - * Wire - connected or broken
- ◆ **Reliability, $R(t)$:**
Probability that the system is up during the **whole** interval $[0, t]$, given it was up at time **0**
- ◆ **Related measure - Mean Time To Failure, MTTF :**
Average time the system remains up before it goes down and has to be repaired or replaced

Traditional Measures - Availability

◆ **Availability, $A(t)$:**

Fraction of time system is up during the interval $[0, t]$

◆ **Point Availability, $A_p(t)$:**

Probability that the system is up at time t

◆ **Long-Term Availability, A :**

$$A = \lim_{t \rightarrow \infty} A(t) = \lim_{t \rightarrow \infty} A_p(t)$$

◆ **Availability is used in systems with recovery/repair**

◆ **Related measures:**

* **Mean Time To Repair, MTTR**

* **Mean Time Between Failures, MTBF = MTTF + MTTR**

$$A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

Need For More Measures

◆ The assumption of the system being in state "up" or "down" is very limiting

◆ **Example:** A processor with one of its several hundreds of millions of gates stuck at logic value 0 and the rest is functional - may affect the output of the processor once in every 25,000 hours of use

◆ The processor is not fault-free, but cannot be defined as being "down"

◆ More detailed measures than the general **reliability** and **availability** are needed

Computational Capacity Measures

Example: N processors in a gracefully degrading system

- ◆ System is useful as long as at least one processor remains operational
- ◆ Let $P_i = \text{Prob} \{i \text{ processors are operational}\}$

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

- ◆ Let c = computational capacity of a processor (e.g., number of fixed-size tasks it can execute)
- ◆ Computational capacity of i processors: $C_i = i \cdot c$
- ◆ Average computational capacity of system:

$$\sum_{i \geq 1} C_i P_i$$

Part.1 .23

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Another Measure - Performability

- ◆ Another approach - consider everything from the perspective of the application
- ◆ Application is used to define "accomplishment levels" L_1, L_2, \dots, L_n
- ◆ Each represents a level of quality of service delivered by the application
- ◆ **Example:** L_i indicates i system crashes during the mission time period T
- ◆ **Performability** is a vector $(P(L_1), P(L_2), \dots, P(L_n))$ where $P(L_i)$ is the probability that the computer functions well enough to permit the application to reach up to accomplishment level L_i

Part.1 .24

Copyright 2007 Koren & Krishna, Morgan-Kaufman

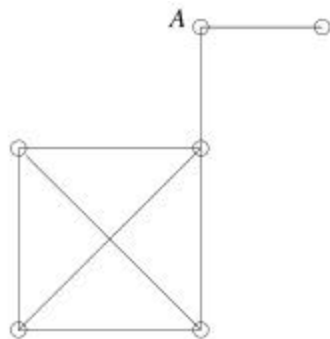
Network Connectivity Measures

- ◆ Focus on the network that connects the processors
- ◆ Classical **Node and Line Connectivity** - the minimum number of nodes and lines, respectively, that have to fail before the network becomes disconnected
- ◆ Measure indicates how vulnerable the network is to disconnection
- ◆ A network disconnected by the failure of just one (critically-positioned) node is potentially more vulnerable than another which requires several nodes to fail before it becomes disconnected

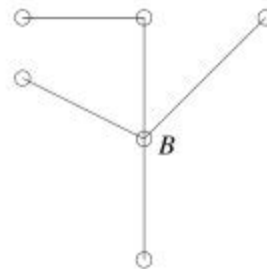
Part.1 .25

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Connectivity - Examples



Network N1



Network N2

Part.1 .26

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Network Resilience Measures

- ◆ Classical connectivity distinguishes between only two network states: connected and disconnected
- ◆ It says nothing about how the network degrades as nodes fail before becoming disconnected
- ◆ Two possible resilience measures:
 - * **Average node-pair distance**
 - * **Network diameter** - maximum node-pair distance
- ◆ Both calculated given probability of node and/or link failure