EDA122/DIT061 Fault-Tolerant Computer Systems

Welcome to Lecture 13

Byzantine failures
Layered fault tolerance

Outline

- Intro to characterization of failure modes
- Byzantine failures
- More on characterization of failure modes
- Layered fault tolerance

Failure modes (from Lecture 2)

- A failure mode describes the nature of a failure, i.e., the way in which a ***service provider*** (a system, subsystem, or module) can fail
- A ***service provider*** can have several failure modes
- Examples of failure modes:
  - ***Value failure*** – a service provider delivers an erroneous result
  - ***Timing failure*** – a service provider delivers a result too late, or too early
  - ***Silent failure*** – a service provider delivers no result
  - ***Signaled failure*** – a service provider sends a failure signal
- A service provider must be equipped with mechanisms for error detection to enforce silent or signaled failures

Value failure vs. Content failure

- A v*alue failure* is the same as a *content failure*
- Both terms are used in the literature
- The term *value failure* is more commonly used
- The term *content failure* is used by Avizienis et al. in “Basic Concepts and Taxonomy of Dependable and Secure Computing”
- We use both terms in the course

Two types of value failures (from Lecture 2)

- **Detectable value failures**: the service user(s) can detect the failure
- **Non-detectable value failure**: the service user(s) cannot detect the failure

Example: Consider a service provider whose outputs are protected by a checksum

Detectable value failure: the output from the service provider has an invalid checksum => a service user can detect the failure by inspecting the checksum

Non-detectable value failure: the value of the output is wrong but the output has a valid checksum => a service user cannot detect the failure by inspecting the checksum

Failure Consistency

- **Consistent failure:** The incorrect service is perceived identically by all users.
- Consistent failures are also known as **Symmetric failures**.
- **Inconsistent failures:** Some or all users perceive the incorrect service differently. Some users may perceive a correct service.
- Inconsistent failures are known as **Byzantine failures** or **Asymmetric failures**.

Characterization of failure modes

- Failure domain
  - Content (Value)
  - Timing
- Detectability of failures
- Consistency of failures
- Consequences of failures on the environment

See “Basic Concepts and Taxonomy of Dependable and Secure Computing”, pp. 18-19.

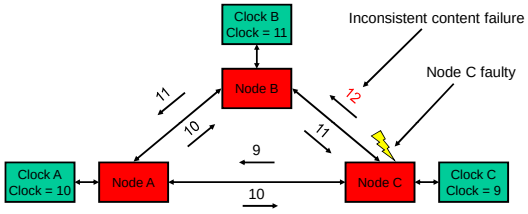
Outline

- Intro to characterization of failure modes
- Byzantine failures
- More on characterization of failure modes
- Layered fault tolerance

Agreement in presence of Byzantine failures

- A Byzantine failure is an inconsistent, non-detectable, content/value failure.
- The Byzantine generals problem:
  - How do replicated units reach agreement on a non-replicated value in the presence of inconsistent content/value failures?
- This problem is also known as the Interactive consistency problem

Impossibility of reaching agreement on clock value



Voting algorithm: Use median value to resynchronize local clock
Node C sends *Clock C = 9* to Node A and *Clock C =12* to Node B
Node A uses *median*{9,10,11} = 10, Node B uses *median*{10,11,12} = 11

Node C is said to be maliciously faulty, since it fools the non-faulty nodes to set their clocks to different values.

Byzantine Generals Problem Scenario

Several armies are camped outside a city which they plan to attack.

Each army is led by a commander. One of the commanders is a general that will issue the order *attack or retreat*.

One or several commanders, including the general, may be traitors.

To win the battle all loyal commanders must attack at the same time.

The traitors will attempt to fool the loyal commanders so that not all of them attack at the same time.

To stop the traitors' malicious plan, all commanders exchange messages directly with each other.

Algorithm ICA(m)

- Tolerates *m* byzantine failures using oral messages (oral message = unprotected message)
- Objectives of the algorithm
  - O1: All fault-free commanders (subordinates) agree on the same command.
  - O2: If the general (the sender) is fault-free, then all the fault-free commanders agree on the command issued by the general.

Algorithm ICA(m)

- Requirements
  - R1: At least *3m+1* units must participate
  - R2: At least *m+1* rounds of communication must take place
  - R3: All units must be synchronized within a known skew of each other
- Assumptions about the message passing system
  - A1: Every message that is sent by a node is delivered correctly by the message passing system to the receiver.
  - A2: The receiver of a message knows which node has sent the message
  - A3: The absence of a message can be detected.

Algorithm ICA(m) (cont'd)

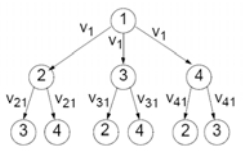
Algorithm ICA(0)

1. The value from the general is sent to every subordinate
2. Each subordinate uses the value received from the general or uses the value RETREAT.

Algorithm ICA(t>0)

1. The value from the general is sent to every subordinate.
2. For each *i*, let v_i be the value that subordinate *i* received from the general, or else be RETREAT if none received. Subordinate *i* acts as the general in Algorithm ICA(*t-1)* to send the value v_i to each of the other (*n-2)* subordinates.
3. For each *i and each j≠i,* let v_j be the value subordinate *i* received from subordinate *j* in step 2 (using Algorithm ICA(t-1)), or else RETREAT if none is received. Subordinate *i* uses the value majority($v_1, v_2, \ldots, v_{n-1}$)

Tolerating one Byzantine failure

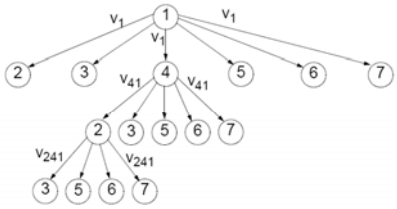


Example: Unit 2 uses *v'1 = majority{v1, v31, v41}*

Number of messages for agreement on one value is: 3 + 2·3 = 9 messages

A majority decision involving 4 units (e.g. 4 sensor values) requires: 9·4 = 36 messages

Tolerating two Byzantine failures



Example: Unit 2 uses *v'1 = majority{v1, v'31, v'41, v'51, v'61,v'71}*
v'41 = majority{v41, v341,v541,v641, v741}

Number of messages for agreement on one value is 6 + 6(5 +5·4) = 156 messages

Byzantine failure in the US Space Shuttle

- A crack (fissure) through a diode appeared in a Space Shuttle while on the launch pad preparing for mission STS-124.
- At 12 minutes and 38 seconds past noon May 13, 2008, this caused a 3-1 split of the four computers that control the Shuttle
- Three seconds later, the split became 2-1-1
- Eventually, the split became 1-1-1-1
- But, none of the processors or their intercommunications were faulty.
- The fault was in a box that sends messages to the computers.

Source and text: Kevin R. Driscoll, Honeywell Aerospace, USA

The US space shuttle's computer system

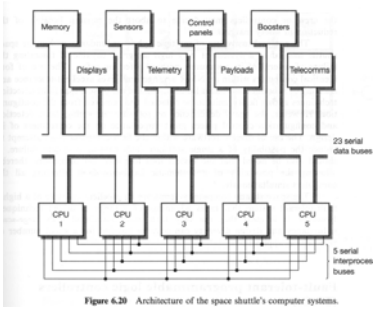


Figure 6.20 Architecture of the space shuttle's computer systems.

Main features of the Space Shuttle Computer

- Consist of five "off-the-shelf" computers with identical hardware
- Four computer executes critical functions in 4MR*
- The computers exchange results via special inter-processor buses and vote on the results in software
- One computer is a backup equipped with an independently developed**, stripped-down version of the critical flight control software
- The fifth computer performs non-critical functions under fault-free circumstances
- The computers send independent commands to actuators, which use hardware voting the mask errors in the commands

* 4MR = NMR, with N=4.
** Design diversity

Cracked diode in US Space Shuttle

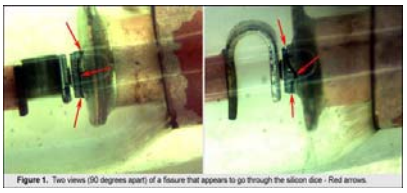
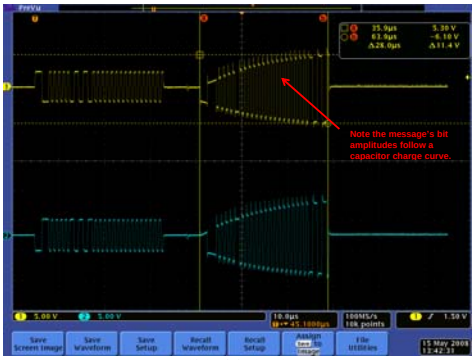


Figure 1. Two views (90 degrees apart) of a fissure that appears to go through the silicon dice - Red arrows.

Normal messages

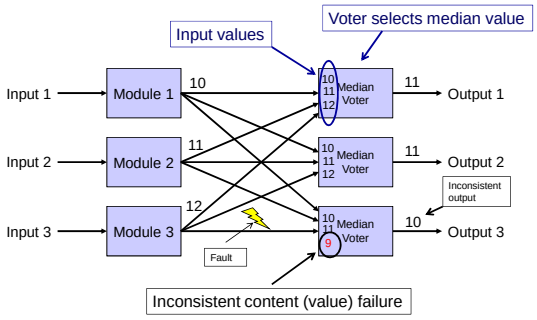
Faulty Message on the Right



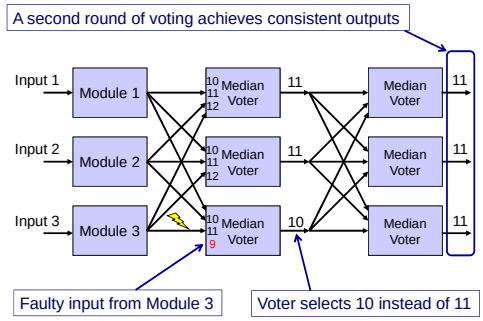
Handling Inconsistent Failures in multi-stage TMR system

- Inconsistent failures can be masked in multi-stage TMR systems through the use of ***multiple voting planes***
- This technique is used in the fly-by-wire system in the JAS Gripen aircraft
- This system will be described in the guest lecture by Lars Holmlund

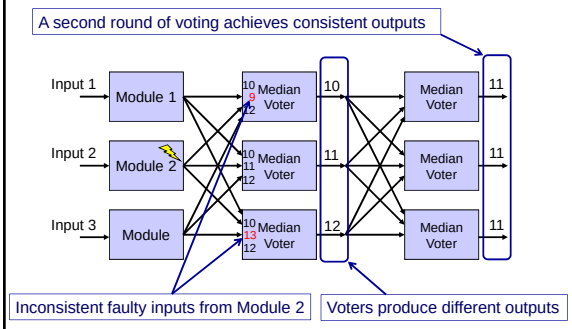
Inconsistent Content Failure in a TMR system



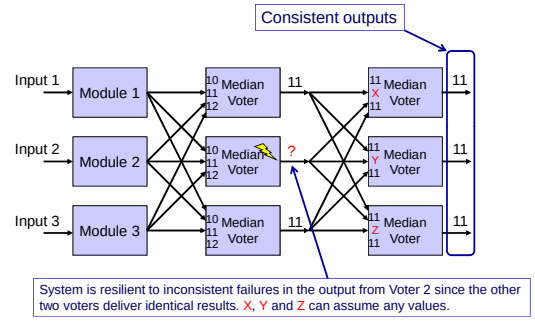
Masking Inconsistent Content Failures with Two Voting Planes



Masking Inconsistent Content Failures with Two Voting Planes



Masking Inconsistent Content Failures with Two Voting Planes



Outline

- Intro to characterization of failure modes
- Byzantine failures
- More on characterization of failure modes
- Layered fault tolerance

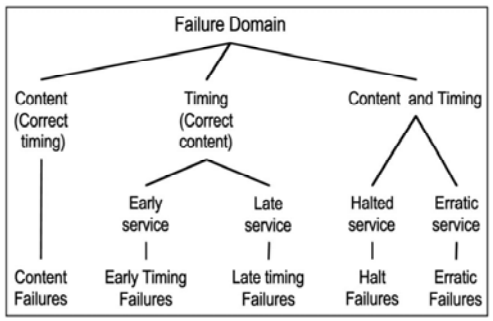
Characterization of Failure Modes

- Failure domain
  - Content (Value)
  - Timing
- Detectability of failures
- Consistency of failures
- Consequences of failures on the environment

See “Basic Concepts and Taxonomy of Dependable and Secure Computing”, pp. 18-19.

Failure domain

- **Content failures:** The content of the information delivered at the service interface deviates from implementing the system function.
- **Timing failures:** The time of arrival or the duration of the information delivered at the service interface (i.e., the timing of service delivery) deviates from implementing the system function.
- **Failure for which both content and timing are incorrect**
  - **Halt failure,** or simply **halt**, when the service is halted; a special case of halt is silent failure, or simply **silence**, when no service at all is delivered at the service interface.
  - **Erratic failure**, when a service is delivered (not halted), but is *erratic* (e.g., babbling)



Failure Detectability

- **Signaled failures –** the system signals service failures to the user(s).
- **Unsignaled failures –** no warning from the system is given
- Signaling at the service interface originates from detection mechanisms checking the correctness of the service.
- The detection mechanisms have themselves two failure modes:
  1. Signaling a failure when no failure has actually occurred; a **false alarm**
  2. Not signaling a service failure; an **unsignaled failure**

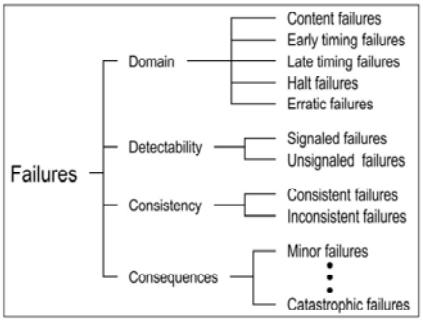
Failure Consistency

- **Consistent failure:** The incorrect service is perceived identically by all users.
- Consistent failures are also known as **Symmetric failures**.
- **Inconsistent failures:** Some or all users perceive the incorrect service differently. Some users may perceive a correct service.
- Inconsistent failures are known as **Byzantine failures** or **Asymmetric failures**.

Consequence of Failures on the Environment

- Grading the *consequences of the failures* upon the system environment involves a definition of *failure severities*
- The definition of the severity levels depends on the application
- Examples of criteria for determining the classes of failure severities include:
  1. the outage duration (for availability)
  2. the possibility of human lives being endangered (for safety)
  3. the type of information that may be unduly disclosed (for confidentiality)
  4. the extent of the corruption of data and the ability to recover from these corruptions (for integrity)

Summary of taxonomy of failure modes



Reading Instructions:

"Basic Concepts and Taxonomy of Dependable and Secure Computing"
(Careful reading)

The following sections shall be read carefully:

2. The Basic Concepts
3.2.1 The Taxonomy of Faults
3.3.1 Service Failures
3.4 Errors
3.5 The Pathology of Failure: Relationship between Faults, Errors, and Failures
4.1 The Definitions of Dependability and Security
5.1 Fault Prevention
5.2 Fault Tolerance
5.3 Fault Removal
5.4 Fault Forecasting

Check out the reading instructions about parts designated as "Normal reading".

Outline

- Intro to characterization of failure modes
- Byzantine failures
- More on characterization of failure modes
- Layered fault tolerance

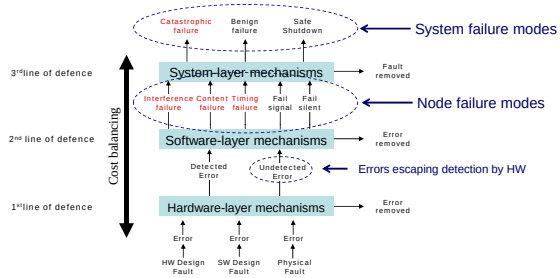
Layered fault tolerance

- Fault-tolerance is typical achieved by combining several mechanisms for error detection, error masking and system recovery located at ***different abstraction layers*** in a computer system
- The division of a system into layers can be done in different ways. We will look at a simple model with three layers:
  - **Hardware layer** – mechanisms implemented in hardware either *within one integrated circuit*, or by *replication of integrated circuits* within a node.
  - **Software layer** – mechanisms implemented in software dealing with errors occurring within a node
  - **System layer** – mechanisms implemented in software dealing with errors occurring in other nodes in the system, or in the system's communication network

Purpose of different layers

- **Hardware layer** – serves as a *first line of defense* that should
  - Correct as many errors as is economically feasible
  - Detect errors that cannot be corrected
- **Software layer** – serves as a *second line of defense* that should
  - Correct errors detected, but ***not corrected by the hardware layer***.
  - Detect errors that are ***not detected or corrected by the hardware layer***
  - Ensure ***appropriate failure semantics*** for the node for any error that cannot be corrected.
- **System layer** – serves as a *third line of defense* that should
  - Detect and correct any errors that are ***not detected or corrected by the software and hardware layers***

Layered fault tolerance



Undesirable failure modes are marked in red.

Overview of Lecture 14

- Trends in hardware reliability
- Error detection
- Read *before the lecture*:
  - Chapter 6.4 in the course book.
  - Designing reliable systems from unreliable components: the challenges of transistor variability and degradation (Background information, not relevant for the exam.)