EDA122/DIT061 Fault-Tolerant Computer Systems
DAT270 Dependable Computer Systems

Welcome to Lecture 6

Hewlett Packard's NonStop System
Ariane 501 Disatster

Outline

- Hardware redundancy
  - Summary of principles
  - Classification of HW redundancy
  - System example: Hewlett Packard's NonStop Computers
- Software redundancy
  - Case study: Ariane 501 Disaster
  - Design diversity
    - N-version programming
    - Recovery blocks

Hardware Redundancy Principles

- Voting redundancy
  - Error masking by majority voting
  - Failure mode assumption for modules: *Non-detectable value failures*
- Standby redundancy
  - Primary module with one or more backup modules
  - Error detection and reconfiguration (fail-over)
  - Failure mode assumption for modules: *Detectable failures*
- Active redundancy
  - Two or more modules operate simultaneously producing replicated results
  - Failure mode assumption for modules:
    - *Silent failures* (fail-silent, fail-stop, fail-fast)
    - *Signaled failures*

Classification of Hardware Redundancy

- **Static Redundancy -** Does *not* require reconfiguration
  - Voting redundancy (requires $2f+1$ units to tolerate f faulty units)
  - Active redundancy (requires $f+1$ units to tolerate f faulty units)
- **Dynamic Redundancy -** Requires reconfiguration
  - Stand-by system (requires $f+1$ units to tolerate f faulty units)
- **Hybrid Redundancy**
  - Combination of static and dynamic redundancy

HP's NonStop Computer Systems

- Highly available multi-processor computers for on-line transaction processing (OLTP) systems
- Typical applications:
  - Automatic teller machines, Stock trading, Funds transfer, 911 emergency centers, Medical records, Travel and hotel reservations, etc
- Availability: 0,99999 – "five nines", or 5 min downtime per year
- Data integrity: 1 FIT = 10^{-9} undetected errors per hour per processor (one undetected data error per billion hours)

Marketing information from HP
(from 2005)

- Telecommunications
  - 135 public telephone companies currently rely on NonStop technology.
  - More than half of all 911 calls in the United States and the majority of wireless calls worldwide depend on NonStop servers.
- Finance
  - Eighty percent of all ATM transactions worldwide and 66 percent of all point-of-sale transactions worldwide are handled by NonStop servers.
  - NonStop technology powers 75 percent of the world's 100 largest electronic funds transfer networks and 106 of the world's 120 stock and commodity exchanges.

NonStop System with self-checked processors

Self-checked processors
- "Fail-fast" (fail-silent) processors
- Stop promptly if an error occurs
- Prevents error propagation

Error detection techniques
- Duplication and comparison
- Error correcting code
- Self-checking logic

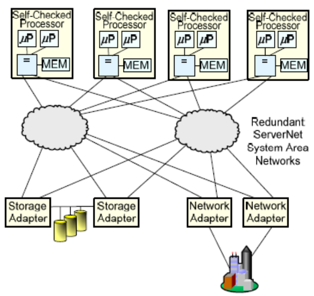


Figure 1: 4 Processor NonStop System with Duplicated and Compared Microprocessors

NonStop System with self-checked processors

Process pairs (standby redundancy)
- Critical software is implemented as a process pair, with one primary and one backup process executing on different processors
- The primary process execute the program and sends state changes regularly to the backup process
- Backup process takes over if the primary process fails by itself or as a result of a processor failure

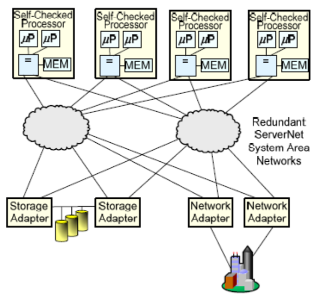


Figure 1: 4 Processor NonStop System with Duplicated and Compared Microprocessors

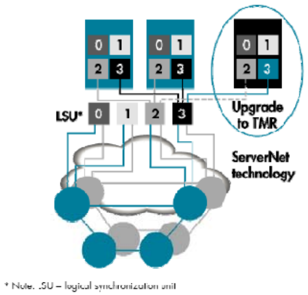
Evolution of self-checked processors

- Self-checking processors (mid 1970's to mid 1980's)
  - Custom designed processors
  - Self-checking logic circuits
  - Memories protected by error correcting codes
- Lock-stepped microprocessors (mid 1980's to late1990's)
  - Two microprocessors run synchronously using the same clock
    - Write operations to the main memory compared
    - A mismatch between memory writes stops the processor
  - Main memory protected by error correcting codes
    - Non-correctable memory errors stops the processor
- Loosely synchronized processors (late 1990's – now)
  - Comparison of I/O-operation (e.g. disk read/write)
  - Dual or triple modular redundancy

Reasons why tightly lock-stepped microprocessors have become infeasible

- Modern microprocessors are nondeterministic - asynchronous events, such as interrupts, can be handled differently by two microprocessors even if they use the same clock
- Power management techniques using variable clock frequencies cannot be used with lock-stepped microprocessors
- Increasing susceptibility to soft errors (radiation induced errors) requires low level error detection and rollback recovery routines, which complicates lock-stepped operation of microprocessors
- Multi-core processors cannot be tightly synchronized

NonStop Advanced Architecture
(released 2005)



- Four Intel Itanium processors per board
- Four logical processors
- The output from the LSU represents a logical processor
- Each logical processor consists of two processors (dual modular redundancy) or three processors (triple modular redundancy)
- TMR allows hardware faults to be masked without fail-over

Logical Processors

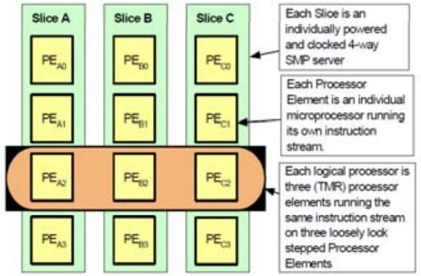


Figure 2: Four Logical NSK Processors built from TMR 4-way SMP servers.

Key Features of The NonStop Advanced Architecture

- Dual or Triple Modular Redundant Servers
- Built from standard 4-way SMP processor modules
- Each processor element (PE) have its own private memory and runs its own private copy of the operating system
- Use a proprietary operating system called NonStop OS that supports process pairs
- Logical processors are formed by two or three processing elements located in different processor modules
- The processors that comprise a logical processor are loosely synchronized by one or two Logical Synchronization Unit (LSU), which also perform voting.
- Critical processes (tasks) can be moved from one logical processor to another logical processor (fail-over)
- Redundant ServerNet SANs (System Area Networks) connect processor modules and I/O devices
- Logical disk volumes are implemented by a pair of mirrored disks

The Itanium Processor

Itanium 2 processor from Intel Corp.

For more info see: http://en.wikipedia.org/wiki/Intel_Itanium

Principles of Fault Tolerance (1)

Fault/Error Containment

- Each processor has its own memory
- Processors communicate via messages passed over the redundant System Area Network (SAN)

No single point of failure

- All system components (processors, I/O adapters, I/O devices, and the System Area Network) are replicated.

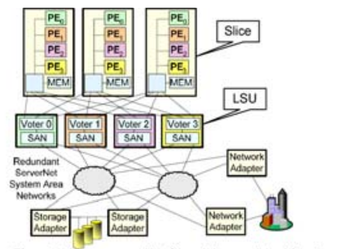


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Principles of Fault Tolerance (2)

Failure mode assumptions

- All system components are self-checking and *fail-fast*
  - Simple errors (e.g., memory access errors or I/O timeouts) can be corrected, or masked, by retry.
  - For other errors, components are *fail-fast*: the components stops when an error is detected (a.k.a. fail silent)

Error detection

- Voting or comparison in the LSU
- ServerNet messages protected by CRC-checksums
- Disk data protected by end-to-end checksum

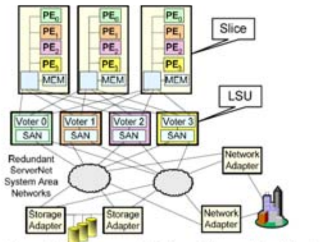


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Protection of disk data

- Disk data is stored on a mirrored pair of disk drives
- For each block of disk data, the processor calculates a checksum covering the data and the block address
- The checksum is written to disk along the with the data
- The processor verifies the checksums when it reads the data from disk
- If the checksum is incorrect, the data is read from the other unit of the mirrored pair

Principles of Fault Tolerance (3)

Dual modular redundancy (DMR)

- Two slices and four LSUs
- A logical processor consists of two processor elements (PEs) and one LSU
- The Logical Synchronization Unit compares the results from the two PEs
- The logical processor stops if the LSU detects a mismatch
- Applications must be configured as a process pair to be fault tolerant.
- A failure of the logical processor initiates fail-over to backup process

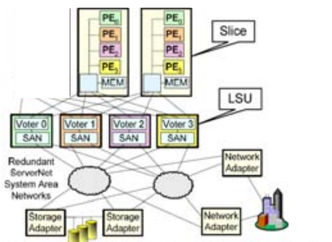


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Principles of Fault Tolerance (4)

Triple modular redundancy (TMR)

- A logical processor consists of three processor elements (PEs) and one or two LSUs
- Voting in the LSU masks processor faults.
- With one LSU, an LSU failure stops the logical processor
- With two LSUs, the system tolerates failures of any two hardware units (PEs or LSUs)
- TMR allows hardware faults to be masked without fail-over
- Reconfigures to DMR if one PE is faulty

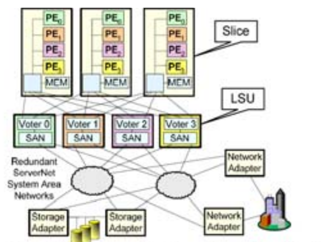


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Outline

- Hardware redundancy
  - Summary of principles
  - Classification of HW redundancy
  - System example: Hewlett Packards's NonStop Computers
- Software redundancy
  - Case study: Ariane 501 Disaster
  - Design diversity
    - N-version programming
    - Recovery blocks
    - System example: Airbus 330/340

Ariane 5 Disaster

"On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in failure. Only about 40 seconds after the initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded."

(From J.L. Lions, et al, ARIANE 5 Flight 501 failure, http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html)

Ariane 5 Disaster

- Development cost ~8 billion euros
- Loss of ~500 million euros
- One year delay of the Ariane program

Ariane 5 Flight Control System

- An Inertial Reference System (SRI) measures the attitude of the launcher and its movements in space. The SRI calculates angles and velocities which are sent to the On-Board Computer (OBC).
- The OBC executes the flight control program and controls the nozzles of the solid boosters and the Vulcain Cryogenic engine.
- There are two SRIs with identical hardware and software operating, one active and one in hot stand-by.

Engine Nozzles

Ariane 5 Disaster

From the press release issued by ESA on July 23, 1996: (http://www.esrin.esa.it/tidc/Press/Press96/press33.html)

The Inquiry Board report begins by presenting the causes of the failure, analysis of the flight data having indicated:

- nominal behaviour of the launcher up to H0 + 36 seconds;
- failure of the back-up Inertial Reference System followed immediately by failure of the active Inertial Reference System;
- swivelling into the extreme position of the nozzles of the two solid boosters and, slightly later, of the Vulcain engine, causing the launcher to veer abruptly;
- self-destruction of the launcher correctly triggered by the rupture of the electrical links between the solid boosters and the core stage.

Ariane 5 Disaster

Chain of events, tracing backwards in time

(From J.L. Lions, et al, ARIANE 5 Flight 501 failure, http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html)

- The launcher started to disintegrate at about H0 + 39 seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher.
- This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine.

Ariane 5 Disaster

Chain of events, tracing backwards in time

- These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the Inertial Reference System 2 (SRI 2). Part of these data at that time did not contain proper flight data, but showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data.
- The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.
- The OBC could not switch to the back-up SRI 1 because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI 2

Ariane 5 Disaster

Chain of events, tracing backwards in time

- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.
- The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.

Ariane 5 Disaster
Chain of events, tracing backwards in time

- The alignment function is operative for 50 seconds after starting of the Flight Mode of the SRIs which occurs at H0 - 3 seconds for Ariane 5. Consequently, when lift-off occurs, the function continues for approx. 40 seconds of flight. This time sequence is based on a requirement of Ariane 4 and is not required for Ariane 5.
- The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time.
- The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.

Lessons Learned from the Ariane 5 Disaster

- Do not expect software, which has proven to be reliable in one environment, to be reliable in another environment
- Ensure that system tests that are realistic
- Use an “intelligent” error handling strategy
  - Consider both *random faults* and *systematic faults*
  - Distinguish between critical services and non-critical services
  - Make critical services resilient to failures

Example of an “intelligent” error handling strategy

- Provide a mechanism to separate *critical services* and *non-critical services*
- Employ a “*never give up*” strategy for critical services
  - Provide error recovery for critical services
  - Make the system resilient to *omission failures* (*temporary service failures*)
  - Provide support for graceful degradation, if possible.
  - Enforce *crash failures* only as a last resort
- Shut down non-critical services that fail

Discussion

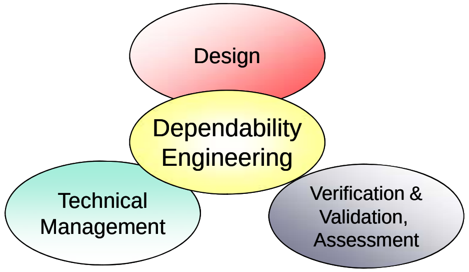
Like many disasters, the Ariane 501 failure was caused by a *lack of knowledge and insight*

In which of the competence areas

- ***Design***
- ***Verification, Validation & Assessment***
- ***Technical Management***

did the Ariane 5 project lack knowledge and insight?

Competence Areas



Recommendations made by the Inquiry Board (1)

- **R1** Switch off the alignment function of the inertial reference system immediately after lift-off. More generally, no software function should run during flight unless it is needed.
- **R2** Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage has to be obtained.
- **R3** Do not allow any sensor, such as the inertial reference system, to stop sending best effort data.
- **R4** Organize, for each item of equipment incorporating software, a specific software qualification review. The Industrial Architect shall take part in these reviews and report on complete system testing performed with the equipment. All restrictions on use of the equipment shall be made explicit for the Review Board. Make all critical software a Configuration Controlled Item (CCI).

Recommendations made by the Inquiry Board (2)

- **R5** Review all flight software (including embedded software), and in particular :
  - Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
  - Verify the range of values taken by any internal or communication variables in the software.
  - Solutions to potential problems in the on-board computer software, paying particular attention to on-board computer switch over, shall be proposed by the project team and reviewed by a group of external experts, who shall report to the on-board computer Qualification Board.
- **R6** Wherever technically feasible, consider confining exceptions to tasks and devise backup capabilities.
- **R7** Provide more data to the telemetry upon failure of any component, so that recovering equipment will be less essential.

Recommendations made by the Inquiry Board (3)

- **R8** Reconsider the definition of critical components, taking failures of software origin into account (particularly single point failures).
- **R9** Include external (to the project) participants when reviewing specifications, code and justification documents. Make sure that these reviews consider the substance of arguments, rather than check that verifications have been made.
- **R10** Include trajectory data in specifications and test requirements.
- **R11** Review the test coverage of existing equipment and extend it where it is deemed necessary.
- **R12** Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent.

Recommendations made by the Inquiry Board (4)

- **R13** Set up a team that will prepare the procedure for qualifying software, propose stringent rules for confirming such qualification, and ascertain that specification, verification and testing of software are of a consistently high quality in the Ariane 5 programme. Including external RAMS experts is to be considered.
- **R14** A more transparent organisation of the cooperation among the partners in the Ariane 5 programme must be considered. Close engineering cooperation, with clear cut authority and responsibility, is needed to achieve system coherence, with simple and clear interfaces between partners.

Software redundancy

Software redundancy techniques can be divided in two major classes:

- With diversity
  - Aim is to tolerate software development faults
  - Design diversity
  - Data diversity
- Without diversity
  - Aim is to handle errors of any origin (physical faults, development faults, operator faults)

What is Software Fault Tolerance?

The term ”software fault tolerance” can mean two things:

1. ”the tolerance of software development faults”, or
2. ”the tolerance of faults by the use of software”

Definition 1 is more commonly used.
Definition 2 is used by N. Storey (author of the course book).

The term ”software redundancy” corresponds to definition 2.

Design Diversity

Design diversity is used to tolerate development faults in hardware and software

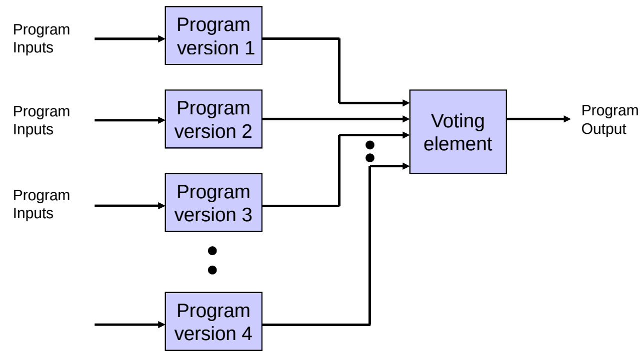
Two techniques for tolerating software design faults:

- N-version programming
- Recovery blocks

N-version programming

- Uses majority voting on results produced by N program versions
- Program versions are developed by different teams of programmers
- Assumes that programs fail independently
- Resembles hardware voting redundancy

N-version programming



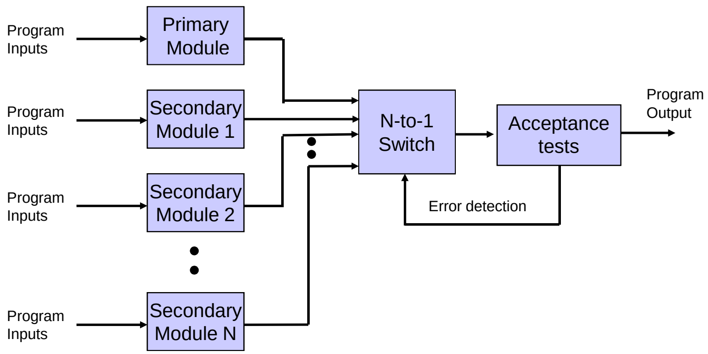
Ensuring independence in N-version programming

- Use different design teams for each version
- Use diverse specifications
- Prevent cooperation among design teams
- Use diverse programming languages, compilers, CASE tools, etc.
- …

Recovery Blocks

- Uses one primary software module and one or several secondary (back-up) software modules
- Assumes that program failures can be detected by acceptance tests
- Executes only the primary module under error-free conditions
- Resembles dynamic hardware redundancy

Recovery blocks



Overview of Lecture 7

- Generalized Stochastic Petri Nets (GSPNs)
- Preparations: Lecture notes

- Design diversity: Airbus flight control system
- Preparations: Section 6.6 and 15.3 in the course book