EDA122 Fault-Tolerant Computer Systems

Welcome to Lecture 12

Experimental studies of software diversity
Study of field failure data

Outline

- Design diversity
  - N-version programming
  - Recovery blocks

Design Diversity

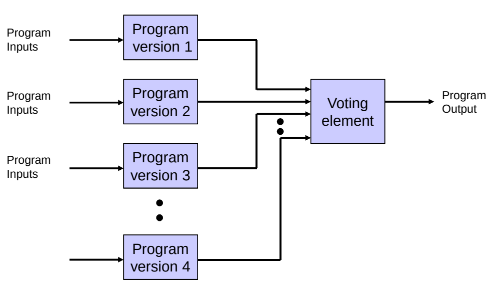
Design diversity is used to tolerate development faults in hardware and software

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

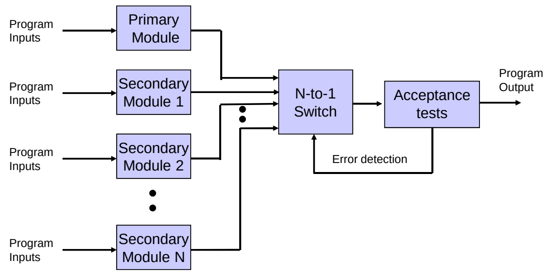
N-version programming



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



Construction of acceptance tests

- An acceptance test is a software implemented check designed to detect errors in the results produced by a primary or a secondary module
- Acceptance tests often relies on application specific information
- An acceptance test is similar to a software assertion (a.k.a. executable assertion).

Comparison of N-version programming and Recovery blocks

N-version programming
- Applied at the program level
- Runs N programs at the same time
- Resembles static hardware redundancy
- Assumes that independence among program versions is achieved by random differences in programming style among programmers

Recovery blocks
- Applied at the module (subprogram) level
- Runs only the primary module under error-free conditions
- Resembles dynamic hardware redundancy
- Independence is achieved by deliberately designing the primary and secondary modules to be as different as possible

Evaluation of N-version programming

Objective
- To investigate if independently developed programs fail independently

Overview
- Missile interceptor program
- 27 versions produced by students at University of Virginia and University of California, Irvine.
- All students was given the same specification
- 200 test cases to validate each program
- 1 million test cases to test independence (simulation of production environment)
- Published 1985

Knight, J.C., N.G. Leveson, and L.D. St. Jean, "A Large Experiment in N-version Programming", Digest of Papers, Int. Symposium on Fault-tolerant Computing (FTCS-15), Ann Arbor, Michigan, June, 1985, pp. 135-139.

Experimental set-up (1)

- 27 versions produced by senior-level students
  - 9 versions from University of Virginia
  - 18 versions from University of California, Irvine
  - Written in Pascal
- Program for anti-missile system
  - Determines if radar reflections represents a incoming hostile missile.
  - Well-known problem – previously used in software engineering experiments.

Experimental set-up (1)

- Input to students
  - Requirements specification
  - Instructed not to cooperate or discuss the problem amongst themselves
  - No restrictions on the use of references
  - 12 input data sets for debugging
- Acceptance test for programs
  - 200 randomly generated tests
  - Different set of tests for each program
  - Resembles testing in real systems
  - Only programs that passed the acceptance test was used in the experimental data

Table 1 – Version Failure Data

Version	Failures	Reliability	Version	Failures	Reliability
1	2	0.999998	15	0	1.000000
2	0	1.000000	16	62	0.999938
3	2297	0.997703	17	269	0.999731
4	0	1.000000	18	115	0.999885
5	0	1.000000	19	264	0.999736
6	1149	0.998851	20	936	0.999064
7	71	0.999929	21	92	0.999908
8	323	0.999677	22	9656	0.990344
9	53	0.999947	23	80	0.999920
10	0	1.000000	24	260	0.999740
11	554	0.999446	25	97	0.999903
12	427	0.999573	26	883	0.999117
13	4	0.999996	27	0	1.000000
14	1368	0.998632			

Evaluation of N-version programming

Occurrence of Multiple Program Failures

# Failed Programs	# Test Cases
2	551
3	343
4	243
5	73
6	32
7	12
8	2

Conclusion: The programs in this experiment do not fail independently*!

(1256 multiple failures, 21257 single failures)

*The hypothesis of independence is rejected at the 99% confidence level.

Table 3 – Correlated Failures Between UVA And UCI

		UVA Versions								
		1	2	3	4	5	6	7	8	9
	10	0	0	0	0	0	0	0	0	0
	11	0	0	58	0	0	2	1	58	0
	12	0	0	1	0	0	0	71	1	0
	13	0	0	0	0	0	0	0	0	0
	14	0	0	28	0	0	3	71	26	0
	15	0	0	0	0	0	0	0	0	0
	16	0	0	0	0	0	1	0	0	0
	17	2	0	95	0	0	0	1	29	0
UCI	18	0	0	2	0	0	1	0	0	0
Versions	19	0	0	1	0	0	0	0	1	0
	20	0	0	325	0	0	3	2	323	0
	21	0	0	0	0	0	0	0	0	0
	22	0	0	52	0	0	15	0	36	2
	23	0	0	72	0	0	0	0	71	0
	24	0	0	0	0	0	0	0	0	0
	25	0	0	94	0	0	0	1	94	0
	26	0	0	115	0	0	5	0	110	0
	27	0	0	0	0	0	0	0	0	0

Discussion (1)

Is it realistic to use students in a software engineering experiment?

- Programming experiences of students outside their degree programs
  - 12 students had less than two years of programming experience
  - 10 students had between two and five years of programming experience
  - 5 students had more than five years of programming experience
- Students had diverse backgrounds

Discussion (2)

Is one million test cases enough?

- Test cases represent “unusal” events.
- “If the program is executed once per second and unusal events occur every ten minutes, then one million test cases correspond to 20 years of operational use”

Conclusions of NVP study (1)

- The assumption of independence of failures among versions ***does not hold***
- The above does not render NVP useless! - It merely shows that the impact of correlated failures must be taken into consideration when estimating the reliability of systems that use NVP.
- The result is only valid for the application used
- Similar results may, or may not, be observed for other applications.

Conclusions of NVP study (2)

- More than half of the software fault was present in two or more programs
- Possible explanations for the high percentage of correlated faults:
  - Programmers make similar mistakes
  - Certain parts of the problem is difficult and lead to mistakes by many programmers
  - Flaws causing uncorrelated failures are easy to catch by normal debugging

Conclusions of NVP study (3)

- Need for further research
  - More experiments needed to draw general conclusions
  - Possible explanations for the high percentage of correlated faults need to be investigated.
  - Relying on random chance to obtain diversity may not be an effective approach. Deliberate diversity may work better.

Evaluation of Recovery Blocks

- Goal: to evaluate recovery blocks for a medium-scale naval command and control system (concurrent real-time system)
- The system provides a simulated radar display overlaid with tracking information. Allows the operator to attack hostile submarines.
- 8000 lines of source code in CORAL, 14 concurrent activities
- Programmed by professional programmers
- Recovery supported by a special recovery cache

Conduct of Experiment

- The command and control system was run against an environment simulator by the operator
- Several typical scenarios were simulated
- Operator logged all abnormal behaviors of the system
- Monitoring routines within the system recorded recovery and failure events

Evaluation of recovery blocks

Naval command and control system (8000 statements in the Coral language)

117 abnormal events

Correct recovery	78 %
Incorrect recovery, program failure	3 %
Incorrect recovery, no program failure	15 %
Unnecessary recovery	3 %

Anderson, T., et al., "Software Fault Tolerance: An Evaluation," IEEE Trans. on Software Engineering, vol. SE-11, no. 12, Dec 1985, pp. 1502-1510.

Overhead for the Case Study

- 60% supplementary development cost
- 33% extra code memory
- 35% extra data memory
- 40% extra execution time

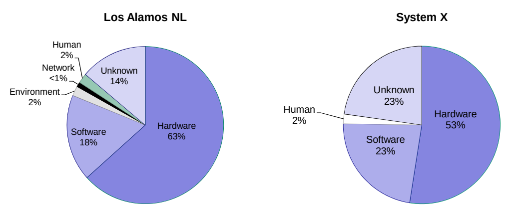
Failure Data from Los Alamos National Laboratories

- Data collected during nine years (1996 – 2005)
- 22 high-performance computing systems
- 4 750 machines
- 24 101 processor
- 23 000 failures
- Covers failures that required interventions by system administrators

Failure data from System X

- Large supercomputing system
- 20 nodes
- 512 processor per node = 10240 processors
- Data covers one year of operation
- Operational since October 2005

Root causes of failures

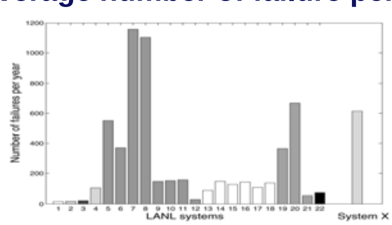


Detailed Root Cause Breakdown of LANL Data

Hardware root causes (%)		Hardware root causes (%) without type E systems		Software root causes (%)		Environmental root causes (%)	
CPU	42.8	Memory Dimm	30.1	Other Software	30.0	Power Outage	48.4
Memory Dimm	21.4	Node Board	16.4	OS	26.0	UPS	21.2
Node Board	6.8	Other	11.8	Parallel File System	11.8	Power Spike	15.1
Other	5.1	Power Supply	9.7	Kernel software	6.0	Chillers	9.8
Power Supply	4.4	Interconnect Interface	6.6	Scheduler Software	4.9	Environment	5.3
Interconnect Interface	3.1	Interconnect Soft Error	3.1	Cluster File System	3.6		
Disk Drive	2.0	CPU	2.4	Resource Mgmt System	3.2		
Interconnect Soft Error	1.3	Fan Assembly	1.8	Network	2.7		
System Board	0.9	Router Board	1.5	User code	2.4		
PCI Backplane	0.8	Fibre Raid Controller	1.4	NFS	1.6		

Important observation: Most outages attributed to memory DIMM:s are caused by transient failures generating more bit flips than the error correcting code can handle.

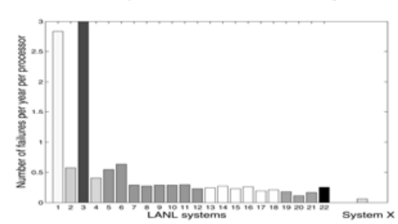
Average number of failure per year



(a)

NOTE: Systems with the same hardware type have the same color.

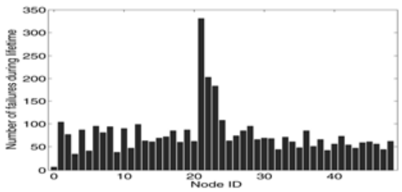
Average number of failures per year normalized by the number of processors



(b)

Observation:
"Failure rates do not grow significantly faster than linearly with system size."

Number of failure per node for system 20

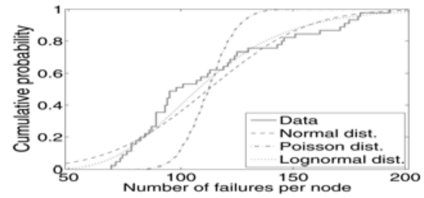


(a)

Observation: The failure rate depend on the workload!
Nodes 21, 22 and 23, which accounts for 20 % of all failures, runs different workloads than the other nodes.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 31

Sampled CDF compared with fitted distributions

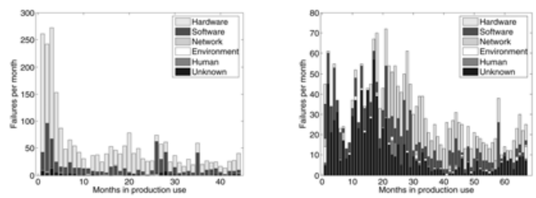


(b)

Observation: Normal and lognormal distributions provide the best fit. The measured data has considerably higher variation that the fitted Poisson distribution. Hence, the Poisson distribution fits poorly with the measured data.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 32

Long term variation of failure rate

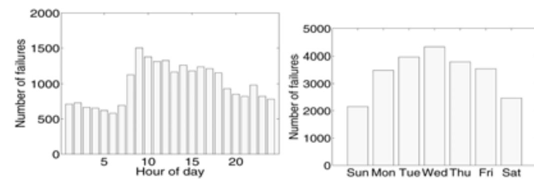


(a) (b)

Observation: Failure rates vary over time, and they do so differently for different systems.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 33

Short term variation of failure rate

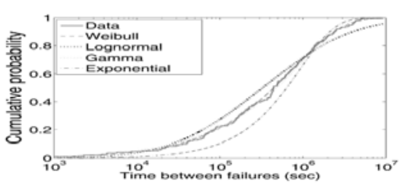


(a) (b)

Important observation: Failure rates depend on the workload of the system.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 34

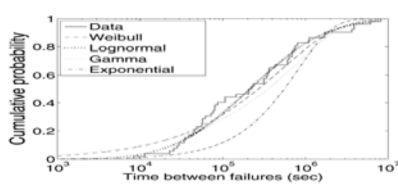
CDF for interarrival times for one node 2000 - 2005



Observation: The Weibull and gamma distributions provides best fit. The squared coefficient of variation C^2 is 1.9 for the measured data.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 35

CDF for interarrival times for one node 1996 - 1999



(a)

Observation: Best fit provided by the lognormal distribution.

Lecture 12 EDA122/DIT061 Fault-Tolerant Computer Systems 36

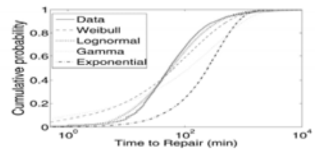
Time to repair

TABLE 4
Statistical Properties of Time to Repair as a Function of the Root Cause of the Failure in the LANL Data

	Unkn.	Hum.	Env.	Netw.	SW	HW	All
Mean (min)	398	163	572	247	369	342	355
Median (min)	32	44	269	70	33	64	54
Std. Dev. (min)	6099	418	808	720	6316	4202	4854
Variability (C^2)	234	6	2	8	293	151	187

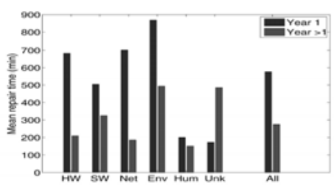
Observation: Note the high values of the squared coefficient of variation C^2.

CDF of repair times



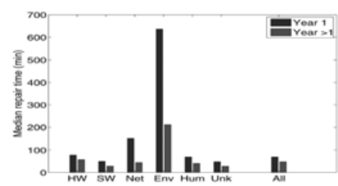
Observation: The lognormal provides the best fit. The exponential distribution is a very poor fit due to the high variability of the repair times.

Effect of learning on mean repair time



Observation: The mean repair time drops after the first year of operation. This reflects the learning curve of the system administrators.

Effect of learning on median repair time



Change of Lectures

- The guest lecture by Lars Holmlund has been moved to October 15.

Overview of Lecture 13

- Byzantine failures
  Read *before the lecture*:
  - Byzantine Agreement, Section 3.1
  - Lecture slides
- Error detection and time redundancy
  Read *before the lecture*:
  - Section 6.3 and 6.4 in the course book
  - Lecture slides