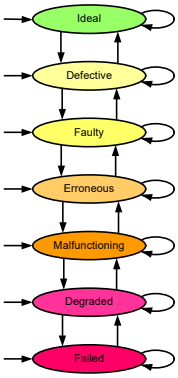
VII Failures: Computational Breaches



"Engineers have learned so well from failures that a major failure today is big news. We no longer expect bridges to collapse or buildings to fall in or spacecraft to explode. Such disasters are perceived as anomalous, . . . We grieve the lost lives, we search among the designers for the guilty. Yet these disasters serve the same function as the failures of an earlier era. Failures remain the engineer's best teacher, his best laboratory."

J. Schallan, reviewing 'To Engineer is Human'

"A man may fall many times, but he won't be a failure until he says that someone pushed him."

Elmer G. Letterman

Chapters in This Part
25. Failure Confinement
26. Failure Recovery
27. Agreement and Adjudication
28. Fail-Safe System Design

A failure occurs when a system's degradation allowance/management capacity has been exceeded in space (due to catastrophic malfunctions) or in time (due to resource exhaustion after a sequence of malfunctions). This is essentially the end of the road and what we have been trying to avoid all along. There is a silver lining, however, in that the larger system, of which the computer is a part, might survive such a failure if adequate provisions are made. Failure confinement techniques, to prevent the spread of damage to data and other system resources, and failure recovery strategies, to return the system to normal operation swiftly, are key elements of the required provisions. We conclude this part of the book by discussing the use of agreement and adjudication schemes to protect data integrity and the design of fail-safe systems to prevent catastrophes.

25 Failure Confinement

"I always turn to the sports page first, which records people's accomplishments. The front page has nothing but man's failures."

Earl Warren

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

Rich Cook

Topics in This Chapter

A common goal of the methods discussed in the preceding chapters is to prevent or postpone computer system failures. In practice, what is actually accomplished is a sharp reduction in the probability, rather than full eradication, of failures. It is thus important to be prepared for failures and to have a good understanding of their underlying causes, consequences, damage confinement and assessment methods, and possible remedial actions. Like other engineering professionals, computer engineers have ethical obligations in reporting problems and following proper design practices to avoid failures. Thus, we conclude the chapter with a review of ethics guidelines for engineers.

25.1 From Failure to Disaster

Computers are components in larger economic, technical, or societal systems. Viewed in this way, a computer failure need not result in insurmountable threats or losses. In many cases, prompt failure detection and activation of back-up systems can avert potential disasters. This strategy is in widespread use for safety-critical systems: jetliners have manual controls and human override options; spacecraft are capable of being controlled from the ground as back-up for their on-board navigation systems; nuclear reactor control systems can be manually bypassed, when needed. Such manual replacement and bypass provisions constitute a buffer between the failed state in our multilevel model and potential disaster (Fig. 25.1).

The provision of manual back-up and bypass capability is a good idea, even for systems that are not safety-critical. On Friday, November 30, 1996 (Thanksgiving weekend), the US railroad company Amtrak lost ticketing capability due to a communication system disruption. Unfortunately for the company, station personnel had no up-to-date fare information as back-up, and were thus unable to issue tickets manually. This lack of foresight led to major coustomer inconvenience and loss of revenue for Amtrak. One must note, however, that in certain cases, such as e-commerce Web sites, manual back-up systems may be impractical.

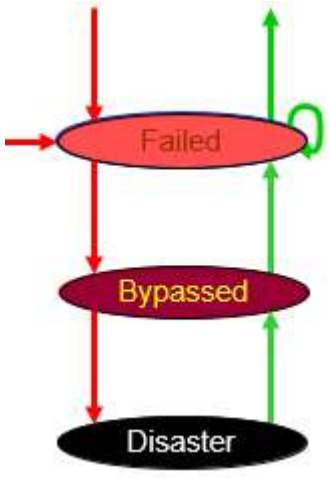


Fig. 25.1 Computer failure may not lead to system failure or disaster.

25.2 Failure Awareness

The first step in proper handling of computer system failures is being aware of their inevitability and, preferably, likelihood. Poring over failure data that are available from experience reports and repositories is helpful to both computer designers and users. System designers can get a sense of where the dependability efforts should be directed and how to avoid common mistakes, while users can become empowered to face failure events and to put in place appropriate recovery plans.

Unfortunately, much of the available failure statistics are incomplete and, at times, misleading. Collecting experimental failure data isn't easy. Widespread experiments are impossible for one-of-a-kind or limited-issue systems and performing them under reasonably uniform conditions is quite a challenge for mass-produced systsme. There is also the embarrassment factor: system operators may be reluctant to report failures that put their technical and administrative skills in doubt, and vendors, especially those who boast about the reliability of their systems, may be financially motivated to hide or obscure failure events. Once a failure event is logged, assigning a cause to it is nontrivial. These caveats notwithstanding, whatever data that is available should be used rather than ignored, perhaps countering any potential bias by drawing information from multiple independent sources.

Importance of collecting experimental failure data

- Indicate where effort is most needed
- Help with verification of analytic models

System outage stats (%)*	Hardware	Software	Operations	Environment
Bellcore [Ali86]	26	30	44	--
Tandem [Gray87]	22	49	15	14
Northern Telecom	19	19	33	28
Japanese Commercial	36	40	11	13
Mainframe users	47	21	16	16
Overall average	30	32	24	14

*Excluding scheduled maintenance

Tandem unscheduled outages

Power	53%
Communication lines	22%
Application software	10%
File system	10%
Hardware	5%

Tandem outages due to hardware

Disk storage	49%
Communications	24%
Processors	18%
Wiring	9%
Spare units	1%

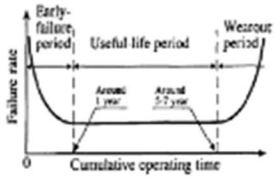
System Failure Data Repositories

Usenix Computer Failure Data Repository
http://usenix.org/cfdr
"The computer failure data repository (CDFR) aims at accelerating research on system reliability by filling the nearly empty collection of public data with detailed failure data from a variety of large production systems. . . .
You first need to register for full access to CFDR data/tools to obtain a user id/password. You can then go to the data overview page."

LANL data, collected 1996-2005: SMPs, Clusters, NUMAs
http://institute.lanl.gov/data/fdata/
From the site's FAQs: "A failure record contains the time when the failure started (start time), the time when it was resolved (end time), the system and node affected, the type of workload running on the node and the root cause."

Memory and Storage Failure Data

Storage failure data: "Disk Failures in the Real World: What does an MTTF of 1,000,000 hours mean to you?" (Schroeder & Gibson, CMU)
http://www.cs.cmu.edu/~bianca/fast07.pdf
From the abstract: ". . . field replacement is a fairly different process than one might predict based on datasheet MTTF."



OBSERVATION 1. *Variance between datasheet MTTF and disk replacement rates in the field was larger than we expected. The weighted average ARR was 3.4 times larger than 0.88%, corresponding to a datasheet MTTF of 1,000,000 Hours.* [Schr07]

Rochester Memory Hardware Error Research Project
http://www.cs.rochester.edu/research/os/memerror/
"Our research focuses on the characteristics of memory hardware errors and their implications on software systems." Both soft errors and hard errors are considered.

Software failure data is available from the following two sources, among others. The Promise Software Engineering Repository [PSER12] contains a collection of publicly available datasets and tools to help researchers who build predictive software models and the software engineering community at large. The failure data at the Software Forensics Center [SFC12] is the largest of its kind in the world and includes specific details about hundreds of projects, with links to thousands of cases.

Let us consider the application of failure data to assessing and validating reliability models through an example.

Example 25.valid: Validating reliability models Consider the reliability state model shown in Fig. 25.disk for mirrored disk pairs, where state *i* corresponds to *i* disks being healthy.

a. Solve the model and derive the disk pair's MTTF, given a disk MTTF of 50 000 hr per manufacturer's claim and an estimated MTTR of 5 hr.

b. In 48 000 years of observation (2 years × 6000 systems × 4 disk pairs / system), 35 double disk failures were logged. Do the observation results confirm the model of part a? Discuss.

Solution: From the MTTF and MTTR values, we find $\lambda = 2 \times 10^{-5}$/hr and $\mu = 0.2$/hr.

a. The model of Fig. 25.disk is readily solved to provide an effective disk pair failure rate of about $2\lambda^2/\mu$ or an approximate disk pair MTTF of $\mu/(2\lambda^2) = 15{,}811$ yr.

b. The observation data suggests a disk pair MTTF of 48 000/35 ≈ 1371 yr. The observed MTTF is more than 11 times worse than the modeled value. The discrepancy may be attributed to one or more of the following factors: exaggerated disk MTTF claim on the part of the manufacturer, underestimation of repair time, imperfect coverage in recovering from the failure of a single disk in the pair. The latter factor can be accounted for by including a transition from state 2 to state 0, with an appropriate rate, in Fig. 25.disk.

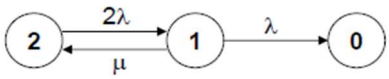


Fig. 25.disk State-space reliability model for a mirrored disk pair.

25.3 Failure and Risk Assessment

Given its importance in our discussion here, let's reproduce equation (2.5.Risk1) and its alternate form, equation (2.5.Risk2), here, giving them new numbers for ready reference in this chapter.

$$\underset{[\textit{consequence / unit time}]}{\text{risk}} = \underset{[\textit{events / unit time}]}{\text{frequency}} \times \underset{[\textit{consequence / event}]}{\text{magnitude}} \qquad (25.3.\text{Risk1})$$

$$\text{risk} = \text{probability} \times \text{severity} \qquad (25.3.\text{Risk2})$$

Like system reliability, the probability or frequency of failures is unknowable. As we derive lower bounds on reliability using pessimistic models, we also obtain upper bounds on failure probability. This is the best we can do, given that overestimating reliability (underestimating the probability of failure) is quite dangerous. Clearly, it is to our advantage to ensure that our reliability lower bound is as close as possible to the actual, unknowable quantity. The more accurate our frequency/probability term in the equations above, the closer our assessed risk will be to the true risk involved.

The magnitude/severity term in the preceding equations is estimated via economic analysis. Before discussing the methods used, we should note that consequences depend on how promptly and thouroughly we handle the failures. Systems must be designed such that failure events are communicated to operators/users via clean, unambiguous messages. Listing the options and the urgency of various actions is a good idea. Two-way communication, that is, adding user feedback to the process, is helpful.

Now, consider the following thought experiment: an attempt to establish how much your life is worth to you.

> You have a 1/10 000 chance of dying today. If it were possible to buy out the risk, how much would you be willing to pay? Assume that you are not limited by current assets, that is, you can use future earnings too.

An answer of \$1000 (risk) combined with the frequency 10^{-4} leads to a magnitude of \$10M, which is the implicit worth you assign to your life. Assigning monetary values to lives, repugnant as it seems, happens all the time in risk assessment. The higher salaries

demanded by workers in certain dangerous lines of work and our willingness to pay the cost of smoke detectors are examples of trade-offs involving the exchange of money for endangering or protecting human lives.

Very Small Probabilities: The Human Factor

Interpretation of data, understanding of probabilities, acceptance of risk

Risk of death / person / year

Influenza	1/5K
Struck by auto	1/20K
Tornado (US MW)	1/455K
Earthquake (CA)	1/588K
Nuclear power plant	1/10M
Meteorite	1/100B

Factors that increase risk of death by $1/10^6$ (deemed acceptable risk)

Smoking 1.4 cigarettes
Drinking 0.5 liter of wine
Biking 10 miles
Driving 300 miles
Flying 1000 miles
Taking a chest X-ray
Eating 100 steaks

US causes of death / 10^6 persons

Auto accident	210
Work accident	150
Homicide	93
Fall	74
Drowning	37
Fire	30
Poisoning	17
Civil aviation	0.8
Tornado	0.4
Bite / sting	0.2

Risk underestimation factors:
Familiarity, being part of our job, remoteness in time or space
Risk overestimation factors:
Scale (1000s killed), proximity

In an eye-opening book [Tale07], author Nassim Nicholas Taleb discusses rare events and how humans are ill-equipped for judging and comparing their probabilities. In a subsequent book [Tale12], that also has important implications for the design of resilient systems, Taleb discusses systems that not only survive disorder and harsh conditions, but thrive and improve in such environment.

25.4 Limiting the Damage

This section to be written based on the following slides.

Prompt failure detection is a prerequisite to failure confinement

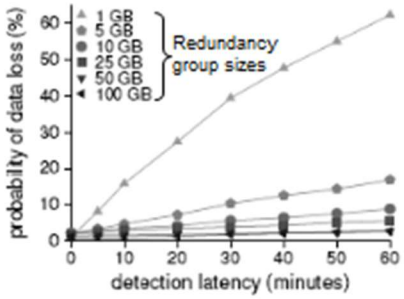
In many cases dealing with mechanical elements, such as wing flaps, reaction time of 10s/100s of milliseconds is adequate (reason: inertia)

In some ways, catastrophic failures that are readily identified may be better than subtle failures that escape detection

Example: For redundant disks with two-way mirroring, detection latency was found to have a significant effect on the probability of data loss

See: http://www.hpdc.org/2004/papers/34.pdf

Failure detection latency can be made negative via "failure prediction" (e.g., in a storage server, increased error rate signals impending failure)



25.5 Failure Avoidance Strategies

There are age-old design principles that engieers should be aware of in order to produce reliable systems. These principles apply to any design, but they are particularly important in developing highly complex hardware and software systems.

– Limit novelty [stick with proven methods and technologies]
– Adopt sweeping simplifications
– Get something simple working soon
– Iteratively add capability
– Give incentives for reporting errors
– Descope [reduce goals/specs] early
– Give control to (and keep it in) a small design team

25.6 Ethical Considerations

Many system failures would not occur if engineers were aware of their ethical responsibilities toward the customers and the society at large. Modern engineering curricula include some formal training in how to deal with ethical quandries. This training is often provided via a required or recommended course that reviews general principles of ethics, discusses them in the context of engineering practice, and provide a number of case studies in which students consider the impact of various career and design decisions. In the author's opinion, just as dependability should not be taught in a course separate from those that deal with engineering design, so too discussion of ethics must be integrated in all courses within an engineering curriculm.

All professional engineering societies have codes of ethics that outline the principles of ethical behavior in the respective professions. For example, the IEEE Code of Ethics [IEEE19] compels us engineers to follow rules of ethics in general (be fair, reject bribery, avoid conflicts of interest) and in technical activities:

> accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
>
> maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
>
> seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;

IEEE has a separate "Code of Conduct" [IEEE14] that spells out in greater details guidelines for responsible practice of engineering.

The Accociation for Computing Machinery is similarly explicit in its ethical recommendations to its members [ACM18]:

> minimize malfunctions by following generally accepted standards for system design and testing;
>
> give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks;

In its comprehensive code of engineering ethics, the National Society of Professional Engineers provides rules of practice (when to accept an assignment, whistle-blowing) and professional obligations (acknowledging errors, being open to suggestions) based on the following six fundamental canons [NSPE18]. According to NSPE, professional engineers:

1. Hold paramount the safety, health, and welfare of the public
2. Perform services only in areas of their competence
3. Issue public statements only in an objective and truthful manner
4. Act for each employer or client as faithful agents or trustees
5. Avoid deceptive acts
6. Conduct themselves honorably, responsibly, ethically, and lawfully

Problems

25.1 MTTF for disk pairs

The solution to Example 25.valid ends with the suggestion that the model can be better matched to reality by including a transition from state 2 to state 0, with an appropriate rate that reflects coverage. Derive the latter rate to bring to the MTTF close to the observed value of 1371 yr.

25.2 Trustworthy artificial intelligence

As AI systems get more powerful and less transparent due to ever-increasing complexity, much discussion is going on about whether or not we can put our trust into machine-learning and other AI systems. Using [EurC19] and [Flor19] as your sources, write a 2-page report that addresses both of the following aspects of the problem.

a. Ethical concerns in delegating critical tasks to autonomous systems.

b. The extent to which we can trust AI with critical decision-making.

25.x Title

Intro

a. xxx

b. xxx

c. xxx

d. xxx

References and Further Readings

[ACM18] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," on-line document, accessed on November 10, 2019. https://www.acm.org/code-of-ethics

[EurC19] European Commision, "Ethics Guideline for Trustworthy AI," High-Level Expert Group on Artificial Intelligence, April 8, 2019. https://ec.europa.eu/futurium/en/ai-alliance-consultation

[Flor19] Floridi, L. and J. Cowls, "A Unified Framework of Five Principles for AI in Society," Harvard Data Science Review, June 14, 2019. https://hdsr.mitpress.mit.edu/pub/l0jsh9d1

[IEEE14] Institute of Electrical and Electronics Engineers, "IEEE Code of Conduct," June 2014, on-line document, accessed on November 10, 2019. https://www.ieee.org/content/dam/ieee-org/ieee/web/org/about/ieee_code_of_conduct.pdf

[IEEE19] Institute of Electrical and Electronics Engineers, "IEEE Code of Ethics," on-line document, accessed on November 10, 2019. https://www.ieee.org/about/corporate/governance/p7-8.html

[LANL12] Los Alamos National Laboratory,

[NSPE18] National Society of Professional Engineers, "Code of Ethics for Engineers," 2018, on-line document, accessed on November 10, 2019. http://www.nspe.org/Ethics/CodeofEthics/index.html

[PSER12] Promise Software Engineering Repository,

[Roch12] Rochester University, "Memory Hardware Error Research Project,"

[Schr07] Schroeder, B. and G. A. Gibson, "Understanding Disk Failure Rates: What Does an MTTF of 1,000,000 Hours Mean to You?" *ACM Trans. Storage*, Vol. 3, No. 3, Article 8, 31 pp., October 2007.

[Schr07a] Schroeder, B. and G. A. Gibson, "Understanding Failures in Petascale Computers," *J. Physics: Conference Series*, Vol. 78, No. 1, Article 012022, 2007.

[Schr09] Schroeder, B. and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *IEEE Trans. Dependable and Secure Systems*, to appear (available online from the *IEEE TDSS* Web site).

[SFC12] Software Forensics Center,

[Siew92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 2nd ed., 1992. Also: A. K. Peters, 1998.

[Tale07] Taleb, N. N., *Black Swan: The Impact of the Highly Improbable*, Random House, 2007.

[Tale12] Taleb, N. N., *Antifragile: Things that Gain from Disorder*, Random House, 2012.

[Usen12] Usenix, "Computer Failure Data Repository,"

[Xin04] Xin, Q., E. L. Miller, and T. J. E. Schwarz, "Evaluation of Distributed Recovery in Large-Scale Storage Systems," *Proc. 13th IEEE Int'l Symp. High Performance Distributed Computing*, pp. 172-181, 2004.

26 Failure Recovery

"The first rule of holes: when you're in one, stop digging."

Molly Ivins

". . . failures appear to be inevitable in the wake of prolonged success, which encourages lower margins of safety. Failures in turn lead to greater safety margins and, hence, new periods of success. To understand what engineering is and what engineers do is to understand how failures can happen and how they can contribute more than successes to advance technology."

Henry Petroski, To Engineer is Human—The Role of Failure in Successful Design

Topics in This Chapter

Planning to deal with computer system failures has a great deal in common with preparations undertaken in anticipation of natural disasters. Since computers are often components in larger control, corporate, or societal systems, interaction with the users and environment must also be factored in. Recovery from computer failures is made possible by systems and procedures for backing up data and programs and for alternate facilities to run applications in case of complete system shut-down due to a catastrophic failure, a natural disaster, or a malicious attack. Once a failure has occurred, investigations may be conducted to establish the cause, assign responsibility, and catalog the event for educational purposes and as part of failure logs to help future designers.

26.1 Planning for Recovery

Just as an organization might hold fire drills to familiarize its personnel with the procedures to be followed in the event of a real fire, so too it must plan for dealing with, and recovering from, computer system failures. Whether an anticipated failure has mild consequences or leads to a disaster, the corresponding recovery procedures must be properly documented and be part of the personnel training programs.

Recovery from a failure can be expressed in the same manner as the recovery block scheme in the program structure 24.5.rb, with the manual or emergency procedure being considered the last alternate and human judgment forming part of the acceptance test. When the failure is judged to be a result of transient environmental conditions, the same alternate may be executed multiple times, before moving on to the next one, including the final initiation of manual recovery.

26.2 Types of Recovery

Many terms have been used to describe the process of recovery from computer system failures. First, systems that are capable of working with diminished resources are referred to as *fail-slow* or *fail-soft*. These terms imply that the system is resilient and won't be brought down when some resources become unavailable. At the opposite extreme, we have *fail-fast* and *fail-hard* systems that are purposely designed to fail quickly and in a physically obvious manner, following the philosophy that subtle failures that may go unnoticed may be more dangerous than overt ones, such as total system shut-down or crash. Extended failure detection latency is undersirable not only owing to potentially errant system behavior, but also because the occurrence of subsequent unrelated failures may overwhelm the system's defenses. Along the same lines, a *fail-stop* system comes to a complete stop rather than behave erratically upon failure. The latter category of systems may be viewed as a special case of a *fail-safe* systems, where the notion of safe behavior generalizes that of halting all actions.

Alongside the terms reviewed in the preceding paragraph, we use *fail-over* to indicate a situation when failure of a unit (such as Web server) is overcome by another unit taking over its workload. This is easily done when the failed unit does not carry much state. If a video is being streamed by a server that fails, the server taking over needs to know only the file name, the receipient's address, and the current point of streaming within the video. Fail-over software is available for Web servers as part of firewalls for most popular operating systems. The term *fail-back* is used to refer to the failed system returning to service, either as the primary unit or as a warm standby. Finally, the term *fail-forward*, derived from the notion of *forward error correction* (the ability of error-correcting codes to let the computation go forward after the occurrence of an error, as opposted to error-detecting schemes leading to backward error correction via rollback to a previous checkpoint) is sometimes, though rarely, used.

26.3 Interfaces and the Human Link

Key elements in ensuring proper human reaction to failure events are the believability and helpfulness of failure warnings.

"No warning system will function effectively if its messages, however logically arrived at, are ignored, disbelieved, or lead to inappropriate actions." Foster, H. D., "Disaster Warning Systems," 1987

Unbelievable failure warnings:
Failure event after numerous false alarms
Real failure occurring in the proximity of a scheduled test run
Users or operators inadequately trained (May 1960 Tsunami in Hilo, Hawaii, killed 61, despite 10-hour advance warning via sirens)

Unhelpful failure warnings:
Autos – "Check engine"
Computer systems – "Fatal error"

Human factors in automated systems

"A few motorists have even driven off a cliff or into oncoming traffic after following [GPS] directions explicitly." [Gree09]

The curse of highly successful automation efforts:
Human operators tend to "switch off" mentally,
so they are not prepared when something unexpected happens
(e.g., commuter train operators text-messaging or dozing off)

A system's human interface not only affects its usability but also contributes heavily to its reliability, availability, and safety. A properly designed and tuned interface can help prevent many human errors that form one of the main sources of system unreliability and risk. The popular saying "The interface is the system" is in fact quite true. Here is a common way of evaluating a user interface. A group of 3-5 usability experts and/or nonexperts judges the interface based on a set of specific criteria. Here are some criteria, which would be used to judge most interfaces [Gree09a]:

- Simplicity: The interface is clean and easy to use.
- Designed with errors in mind: The interface assumes that the user will make errors. Errors are avoidable (via the requirement for confirmation of critical actions) and easy to reverse.

- Visibility of system state: The user knows about what is happening inside the computer from looking at the interface.
- Speaks the user's language: Uses concepts that are familiar to users. If there are different user classes (say, novices and experts in the field), the interface is understandable to all.
- Minimizes human memory load: Human memory is fallible and people are likely to make errors if they must remember information. Where possible, critical information appears on the screen. Recognition and selection from a list are easier than memory recall.
- Provides feedback to the user: When the user acts, the interface confirms that something happened. The feedback may range from a simple beep to indicate that a button press was recorded to a detailed message that describes the consequences of the action.
- Provides good error messages: When errors occur, the user is given helpful information about the problem. Poor error messages can be disastrous.
- Consistency: Similar actions produce similar results. Visually similar objects (colors, shapes) are related in an important way. Conversely, objects that are fundamentally different have distinct visual appearance.

When an interface is implicated in an accident, the most common problems are inconsistency, hiding of the system state, and failure to design for error.

26.4 Backup Systems and Processes

Backing up of data is a routine practice for corporate and enterprise systems. The data banks held by a company, be they data related to designs, inventory, personnel, suppliers, or customer base, is quite valuable to the company's business and must be rigorously protected against loss. Unfortunately, many personal computer users do not take data backup seriously and end up losing valuable data as a result. Here are some simple steps that nonexpert computer users can take to protect their data files:

Use removable storage: external hard drive, flash drive, CD/DVD
E-mail file copy to yourself (not usable for very large files)
Run a backup utility periodically (full or incremental backup)
Subscribe to an on-line backup service (plain or encrypted)
Do not overwrite updated files, but create new versions of them

[Elaborate on the importance of on-site and off-site backups.]

26.5 Blame Assessment and Liability

In cases where the cause of a failure isn't immediately evident, or when multiple parties involved do not agree on the cause, computer forensics may come into play. Computer forensics is a relatively new speciality of increasingly utility in:

Scanning computers belonging to defendants or litigants in legal cases
Gathering evidence against those suspected of wrongdoing
Learning about a system to debug, optimize, or reverse-engineer it
Assessing compromised computer systems to determine intrusion method
Analyzing failed computer systems to determine cause of failure
Recovering data after computer failures

There are several journals on computer forensics, digital investigations, and e-discovery.

26.6 Learning from Failures

As we discussed in Section 1.4 in connection with our multilevel model of dependable computing, a system may fail when it degrades beyond its degradation management capacity (downward transition) or it may begin its life in the failed state (sideway transition). A system of the latter kind never gets off the ground and its design may be scrapped before it becomes operational. The following are a few examples of such failed systems, along with the cost of the project to build them:

Automated reservations, ticketing, flight scheduling, fuel delivery, kitchens, and general administration, United Airlines + Univac,
started 1966, target 1968, scrapped 1970, $50M

Hotel reservations linked with airline and car rental, Hilton + Marriott + Budget + American Airlines, started 1988, scrapped 1992, $125M

IBM workplace OS for PPC (Mach 3.0 + binary compatibility with AIX + DOS, Mac OS, OS/400 + new clock mgmt + new RPC + new I/O + new CPU), started 1991, scrapped 1996, $2B

US FAA's Advanced Automation System (to replace a 1972 system), started 1982, scrapped 1994, $6B

London Ambulance Dispatch Service, started 1991, scrapped 1992,
20 lives lost in 2 days, $2.5M

Many more such failures exist, as exemplified by the following partial list:

- Portland, Oregon, Water Bureau, $30M, 2002
- Washington D.C., Payroll system, $34M, 2002
- Southwick air traffic control system $1.6B, 2002
- Sobey's grocery inventory, Canada, $50M, 2002
- King County financial mgmt system, $38M, 2000
- Australian submarine control system, 100M, 1999
- California lottery system, $52M
- Hamburg police computer system, 70M, 1998

- Kuala Lumpur total airport management system, $200M, 1998
- UK Dept. of Employment tracking, $72M, 1994
- Bank of America Masternet accounting system, $83M, 1988
- FBI virtual case, 2004
- FBI Sentinel case management software, 2006

Learning from failures is one of the tenets of engineering practice. As elegantly pointed out by Henry Petroski [Petr06]:

> "When a complex system succeeds, that success masks its proximity to failure. . . . Thus, the failure of the *Titanic* contributed much more to the design of safe ocean liners than would have her success. That is the paradox of engineering and design."

Problems

26.1 Resilience engineering

According to a published discussion [Kris12], companies with critical computing infrastructures have started to adopt resilience engineering, a practice that has been common in other high-risk industries such as aviation and health care. In a maximum of one typed page, describe the key notions of resilience engineering (~1/2 page), Amazon's adopted strategy (~1/4 page), and Google's approach to it (~1/4 page).

26.x Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

References and Further Readings

[Bain87] Bainbridge, L. "Ironies of Automation: Increasing Levels of Automation Can Increase, Rather than Decrease, the Problems of Supporting the Human Operator," in *New Technology and Human Error*, J. Rasmussen, K. Duncan, and J. Leplat (eds.), pp. 276–283, Wiley, 1987.

[Dekk06] Dekker, S., *The Field Guide to Understanding Human Error*, Ashgate, 2006.

[Gree09] Greengard, S., "Making Automation Work," *Communications of the ACM*, Vol. , No. 12, pp. , December 2009.

[Gree09a] Green, M., "Error and Injury in Computers & Medical Devices," Web page accessed on November 29, 2012. http://www.visualexpert.com/Resources/compneg.html

[Kris12] Krishnan, K., "Practice: Weathering the Unexpected," *Communications of the ACM*, Vol. 55, No. 11, pp. 48-52, November 2012.

[Petr06] Petroski, H., *Success Through Failure: The Paradox of Design*, Princeton Univ. Press, 2006, p. 95.

27 Agreement and Adjudication

"There is nothing more likely to start disagreement among people or countries than an agreement."

E. B. White

"We judge ourselves by what we feel capable of doing, while others judge us by what we have already done."

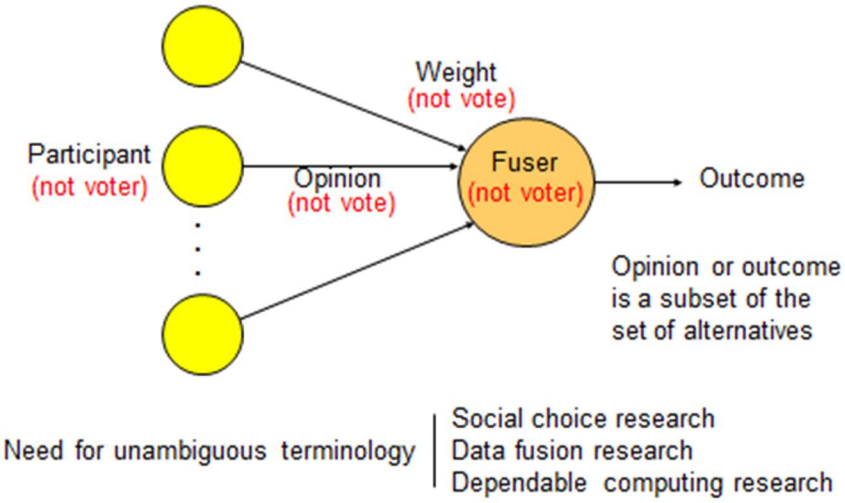
Henry Wadsworth Longfellow

Topics in This Chapter

In Chapter 12, we studied simple voting schemes and their associated hardware implementations. In this chapter, we introduce a variety of more flexible, and thus computationally more complex, voting schemes that are often implemented in software or a combination of hardware and software. Voting schemes constitute particular instances of a process known as data fusion, where suspect or incomplete data from various sources are used to derive more accurate or trustworthy values. One complication with voting or data fusion in a distributed environment is the possibility of communication errors and Byzantine (absolute worst-case) failures. Discussions of these notions conclude this chapter.

27.1 Voting and Data Fusion

This section to be written based on the following slides.



Introduction to Voting

Voting schemes and associated terminology in dependable computing were originally derived from concepts in sociopolitical elections

With inputs drawn from a small set of integers, the similarity between the two domains is strong

Example: Radar image analysis used to classify approaching aircraft type as civilian (0), fighter (1), bomber (2).

If three independent units arrive at the conclusions ⟨1, 1, 2⟩, then the presence of a fighter plane may be assumed

Option or candidate 1 "wins" or garners a majority

With a large or infinite input domain, voting takes on a new meaning

Example: There is no strict majority when distance of an approaching aircraft, in km, is indicated as ⟨12.5, 12.6, 14.0⟩, even though the good agreement between 12.5 and 12.6 could lead to a 12.55 km estimate

Role of Voting Units in Dependable Computing

John von Neumann, 1956: "Probabilistic Logic and Synthesis of Reliable Organisms from Unreliable Components"

Hardware voting for multichannel computation

High performance (pipelined)

TMR, NMR

Software voting for multiversion programming

Imprecise results (approximate voting)

NVP, SIFT

Consistency of replicated data

Weighted voting and weight assignment

Quorum, Consensus

Voting schemes in these three contexts share some properties

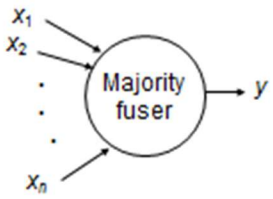
General schemes have been devised to cover all these instances

What We Already Know About Voting

Majority voting: Select the value that appears on at least $\lfloor n/2 \rfloor + 1$ of the n inputs

Number n of inputs is usually odd, but does not have to be

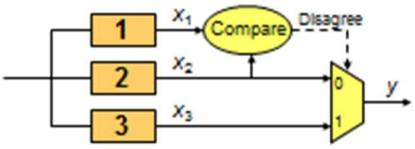
Example: vote(1, 2, 3, 2, 2) = 2



Majority fusers can be realized by means of comparators and muxes

This design assumes that in the case of 3-way disagreement any one of the inputs can be chosen

Can add logic to this fuser so that it signals three-way disagreement



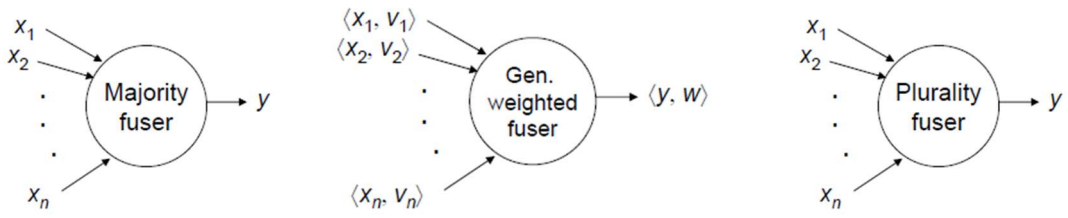
27.2 Weighted Voting

From our discussion of hardware voting in Section xxxx, we are already familiar with the notion of majority voting. A majority fuser sets the output y to be the value provided by a majority of the inputs x_i, if such a majority exists (Fig. 27.fuser-a). We also know that majority fusers can be built from comparators and multiplexers.

Weighted fusion, which covers majority fusion as a special case, can be defined as follows. Given n input data objects $x_1, x_2, \ldots, x_n$ and associated nonnegative real weights $v_1, v_2, \ldots, v_n$, with $\Sigma v_i = V$, compute output y and its weight w such that y is "supported by" a set of input objects with weights totaling w, where w satisfies a condition associated with a chosen subscheme. Here are some subschemes that can be used in connection with the general arrangement of Fig. 27-fuser-b:

Unanimity	$w = V$
Majority	$w > V/2$
Supermajority	$w \geq 2V/3$
Byzantine	$w > 2V/3$
Plurality	(w for y) $\geq$ (w for any $z \neq y$)
Threshold	$w >$ some preset lower bound

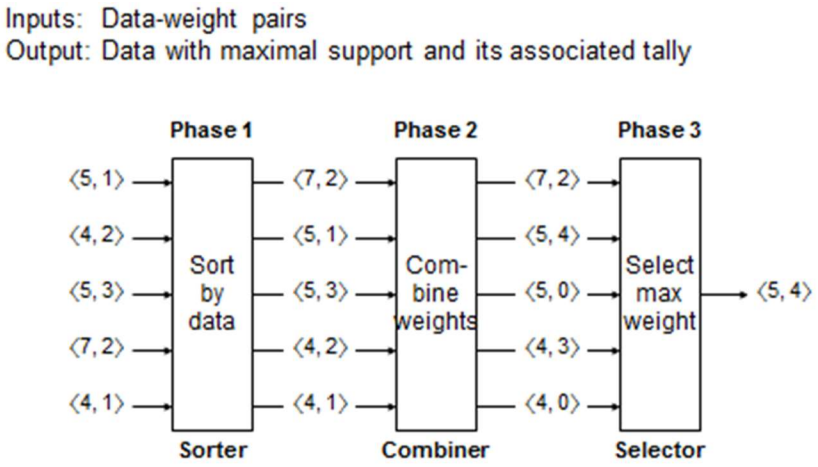
Plurality fusion (in its special nonweighted case depicted in Fig. 27.fuser-c) selects the value that appears on the largest number of inputs and presents it at output. With the input values {1, 3, 2, 3, 4}, the output will be 3. It is interesting to note that with approximate values, selection of the plurality results may be nontrivial. If the inputs are {1.00, 3.00, 0.99, 3:00, 1.01}, one can legitimately argue that 1.00 constitutes the proper plurality result. We will discuss approximate voting in more detail later. For now, we note that median fusion would have produced the output 1.00 in the latter example.



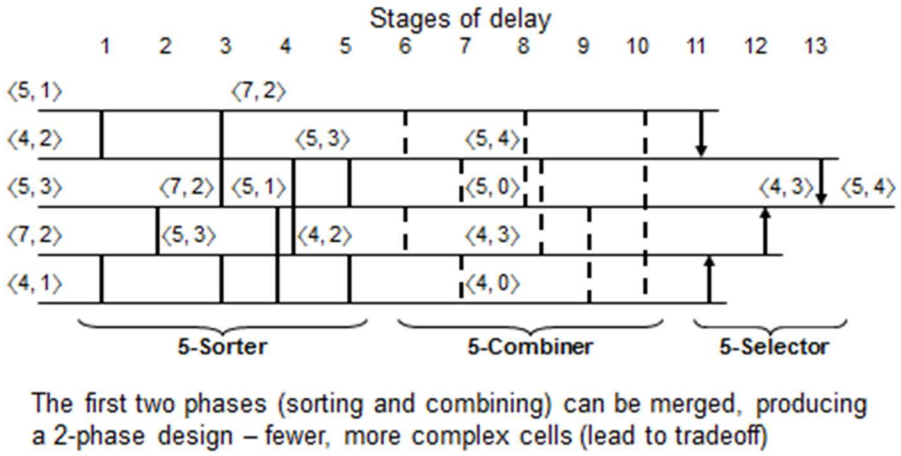
(a) Nonweighted majority (b) Generalized weighted (c) Nonweighted plurality

Fig. 27.fuser Three kinds of data fusers derived from voting schemes.

Implementing weighted plurality voting units [Parh91].



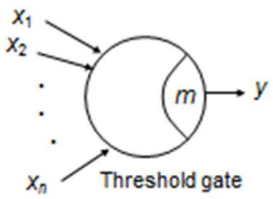
Details of a sorting-based plurality voting unit [Parh91].



The first two phases (sorting and combining) can be merged, producing a 2-phase design – fewer, more complex cells (lead to tradeoff)

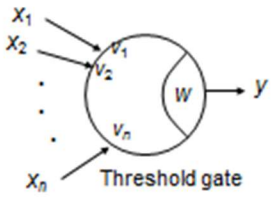
Threshold voting and its generalizations

Simple threshold (m-out-of-n) voting:
Output is 1 if at least m of the n inputs are 1s



Majority voting is a special case of threshold voting: ($\lfloor n/2 \rfloor + 1$)-out-of-n voting

Weighted threshold (w-out-of-Σv_i) voting:
Output is 1 if $\Sigma v_i x_i$ is w or more



Agreement or quorum sets
$\{x_1, x_2\}$, $\{x_2, x_3\}$, $\{x_3, x_1\}$ – same as 2-out-of-3
$\{x_1, x_2\}$, $\{x_1, x_3, x_4\}$, $\{x_2, x_3, x_4\}$

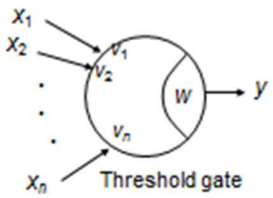
The 2nd example above is weighted voting with $v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$

Agreement sets are more general than weighted voting in the sense of some agreement sets not being realizable as weighted voting

Usefulness of weighted threshold voting

Unequal weights allow us to take different levels of input reliabilities into account

Zero weights can be used to disable or purge some of the inputs (combined switching-voting)



Maximum-likelihood voting
prob{x_1 correct} = 0.9
prob{x_2 correct} = 0.9
prob{x_3 correct} = 0.8
prob{x_4 correct} = 0.7
prob{x_5 correct} = 0.6

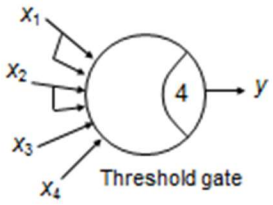
Max-likelihood voting can be implemented in hardware via table lookup or approximated by weighted threshold voting (otherwise, we need software)

Assume $x_1 = x_3 = a$, $x_2 = x_4 = x_5 = b$

prob{a correct} = 0.9 × 0.1 × 0.8 × 0.3 × 0.4 = 0.00864
prob{b correct} = 0.1 × 0.9 × 0.2 × 0.7 × 0.6 = 0.00756

Implementing weighted threshold voting units

Ex.: Implement a 4-input threshold voting unit with $v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$



Strategy 1: If weights are small integers, fan-out each input an appropriate number of times and use a simple threshold voting unit

Strategy 2: Use table lookup based on comparison results

x_1=x_2	x_1=x_3	x_2=x_3	x_3=x_4	Result
1	x	x	x	x_1
0	0	0	1	Error
0	1	0	1	x_1
0	0	1	1	x_2
0	x	x	0	Error

Is this table complete? Why, or why not?

Strategy 3: Convert the problem to agreement sets (discussed next)

27.3 Voting with Agreement Sets

An agreement set for an n-input voting scheme is a subset of the n inputs so that if all inputs in the subset are in agreement, then the output of the voting scheme is based on that particular subset. A voting scheme can be fully characterized by its agreement sets. For example, simple 2-out-of-3 majority voting has the agreement sets $\{x_1, x_2\}$, $\{x_2, x_3\}$, and $\{x_3, x_1\}$. Clearly, the agreement sets cannot be arbitrary if the voting outcome is to be well-defined. For example, in a 4-input voting scheme, the agreement sets cannot include $\{x_1, x_2\}$ and $\{x_3, x_4\}$. It is readily proven that for a collection of agreement sets to make sense, no two sets should have an empty intersection.

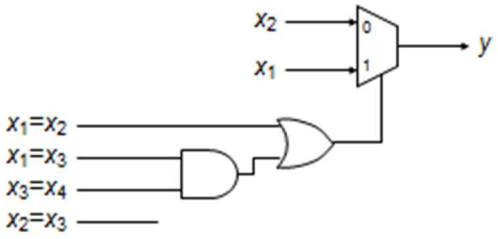
Implementing agreement-set voting units

Example: Implement a voting unit corresponding to the agreement sets $\{x_1, x_2\}$, $\{x_1, x_3, x_4\}$, $\{x_2, x_3, x_4\}$

Strategy 1: Implement as weighted threshold voting unit, if possible

Strategy 2: Implement directly

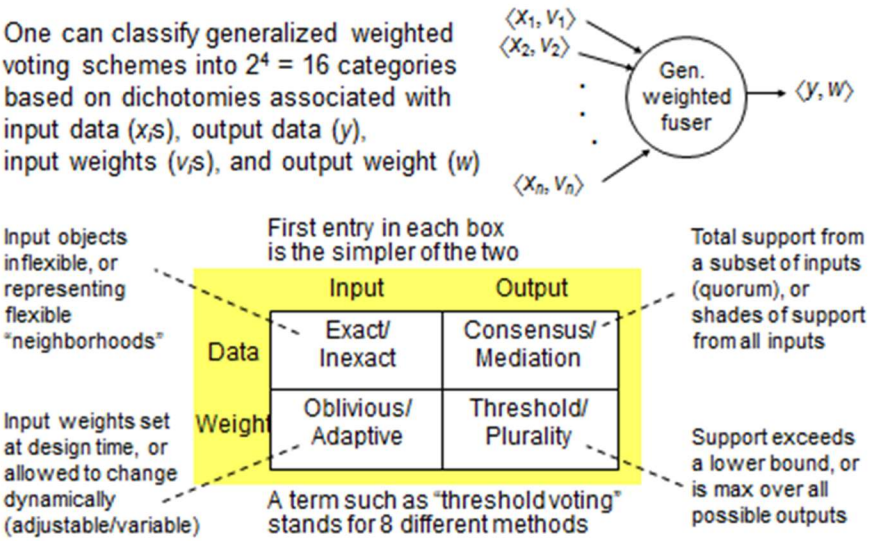
Find a minimal set of comparators that determine the agreement set



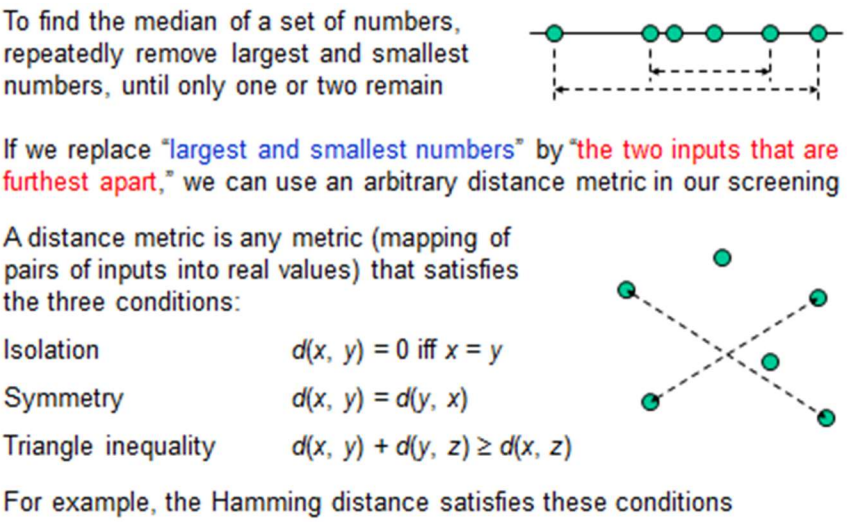
Complete this design by producing the "no agreement" signal

27.4 Variations in Voting

This section to be written based on the following slides.



Generalized median voting

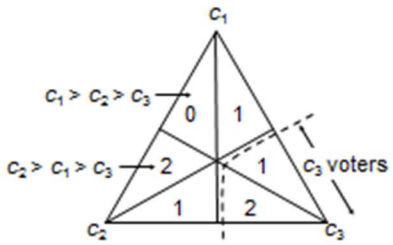


The impossibility of perfect voting

Properties of an ideal voting scheme:

1. No big brother
   (participants free to express preferences)
2. Independence of irrelevant alternatives
   (preference for one candidate over another is independent of all others)
3. Involvement
   (every outcome is possible)
4. No dictatorship or antidictatorship
   (outcome not always conforming to, or opposite of, one participant's view)

Arrow's Theorem:
No voting scheme exists that satisfies all four conditions



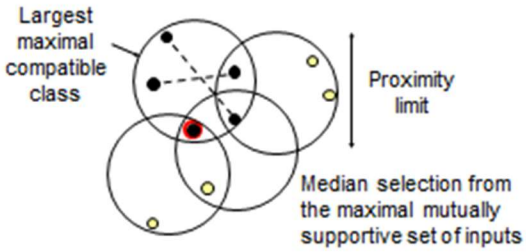
True majority voting scheme:
Each participant orders all the candidates; no circular preference allowed
Choose a candidate who beats every other one in pairwise competitions
(both simple majority and plurality rules fail to choose a candidate)

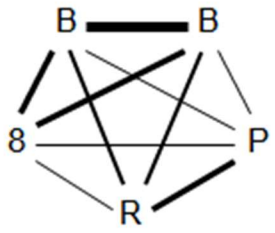
Approximate voting

The notion of an input object "supporting" a particular output (akin to a hypothesis supporting an end result or conclusion) allows us to treat approximate and exact voting in the same way

Example 1: Input objects are points in the 2D space and the level of "support" between them is a function of their Euclidean distance

Example 2: Input objects are conclusions of character recognizers as to the identity of a character, with varying degrees of mutual support

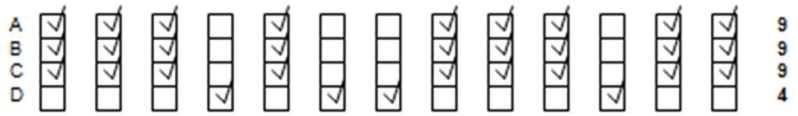




Approval voting

Approval voting was introduced to prevent the splitting of votes among several highly qualified candidates from leading to the election of a less qualified candidate in plurality voting

In approval voting, a participant divides the candidates into two subsets of "qualified" and "not qualified" and indicates approval of the first subset



In the context of computing, approval voting is useful when a question has multiple answers or when the solution process is imprecise or fuzzy

Example question: What is a safe setting for a particular parameter in a process control system?

When the set of approved values constitute a continuous interval of real values, we have "interval" inputs and "interval" voting

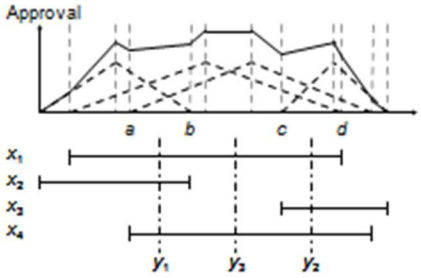
Interval voting

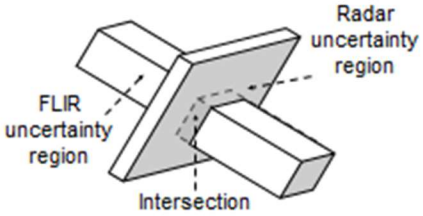
Inputs to the voting process are intervals, representing approved values

How should the voting result be derived from the input intervals?

Various combining rules can be envisaged
If there is overlap among all intervals, then the decision is simple

Depending on context, it may make sense to consider greater levels of approval near the middle of each interval or to associate negative approval levels outside the approved intervals





Combining two 3D intervals

27.5 Distributed Agreement

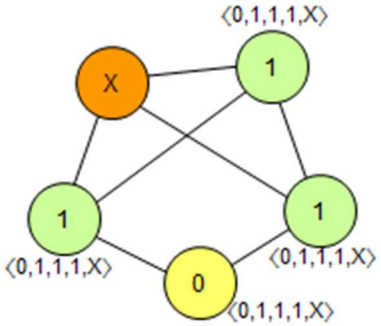
This section to be written based on the following slides.

Problem: Derive a highly reliable value from multiple computation results or stored data replicas at multiple sites

Key challenge: Exchange data among nodes so that all healthy nodes end up with the same set of values; this guarantees that running the same decision process on the healthy nodes produces the same result

Errors are possible in both data values and in their transmission between sites

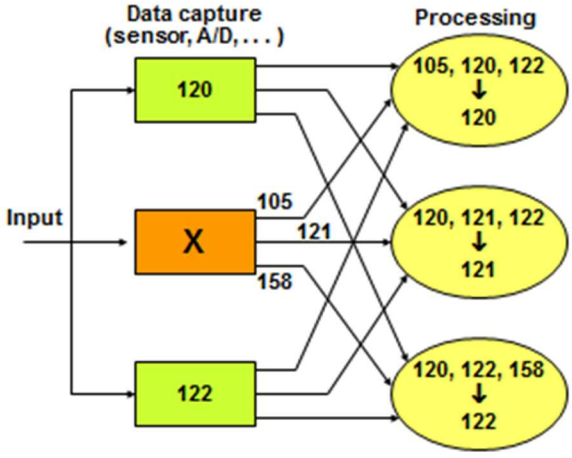
Agreement algorithms generally use multiple rounds of communication, with values held at each site compared and filtered, until the set of values held at all sites converge to the same set



Byzantine failures in distributed voting

Three sites are to collect three versions of some parameter and arrive at consistent voting results

Assume median voting



The interactive consistency (IC) algorithm

ICA(0) [no failure]

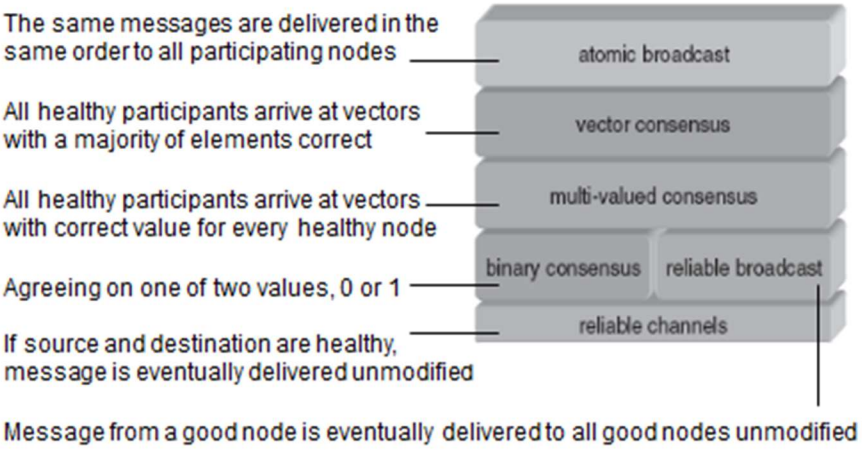
1. The transmitter sends its value to all other $n - 1$ nodes
2. Each node uses the value received from the transmitter, or a default value Φ if it received no value

ICA(f), $f > 0$ [f failures]

1. The transmitter sends its value to all other $n - 1$ nodes
2. Let v_i be the value received by node i from the transmitter, or a default value Φ if it received no value; node i then becomes the transmitter in its own version of ICA($f - 1$), sending its value to $n - 2$ nodes
3. For each node i, let $v_{i,j}$ be the value it received from node j, or a default value Φ if it received no value from node j. Node i then uses the value majority($v_{i,1}, v_{i,2}, \ldots, v_{i,i-1}, v_{i,i+1}, \ldots, v_{i,n}$)

$O(n^{f+1})$ messages needed, in $f + 1$ rounds, to tolerate f Byzantine failures

Building upon consensus protocols



Source: M. Correia, N. Ferreira Neves, P. Veríssimo, "From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures," *The Computer J.*, 2005.

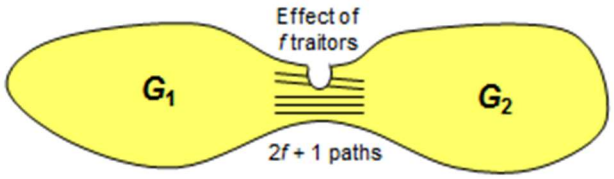
Correctness and performance of ICA

Theorem 1: With ICA(f), all nonfailed nodes will agree on a common value, provided that $n \geq 3f + 1$ (proof is by induction on f)

ICA works correctly, but it needs an exponential number of messages:
$(n-1) + (n-1)(n-2) + (n-1)(n-2)(n-3) + \ldots + (n-1)(n-2) \ldots (n-m)$

More efficient agreement algorithms exist, but they are more difficult to describe or to prove correct; $f + 1$ rounds of message exchange is the least possible, so some algorithms trade off rounds for # of messages

Theorem 2: In a network G with f failed nodes, agreement is possible only if the connectivity is at least $2f + 1$



27.6 Byzantine Resiliency

The Byzantine generals problem

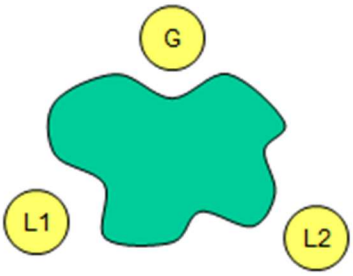
A general and $n - 1$ lieutenants lead n divisions of the Byzantine army camped on the outskirts of an enemy city

The n divisions can only communicate via messengers

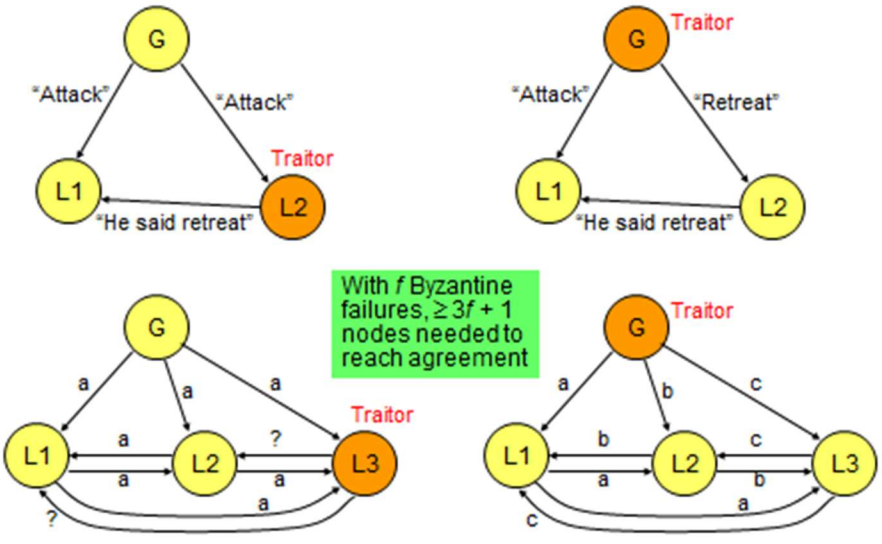
We need a scheme for the generals to agree on a common plan of action (attack or retreat), even if some of the generals are traitors who will do anything to prevent loyal generals from reaching agreement

The problem is nontrivial even if messengers are totally reliable

With unreliable messengers, the problem is very complex



Byzantine generals with reliable messengers



Bounds for Bzantine resiliency:

To tolerate f Byzantine failures:

We need $3f + 1$ or more FCRs (fault containment regions)

FCRs must be interconnected via at least $2f + 1$ disjoint paths

Inputs must be exchanged in at least $f + 1$ rounds

Corollary 1: Simple 3-way majority voting is not Byzantine resilient

Corollary 2: Because we need $2f + 1$ good nodes out of a total of $3f + 1$ nodes, a fraction $(2f + 1)/(3f + 1) = 2/3 + 1/(9f + 3)$ of the nodes must be healthy
This is greater than a supermajority (2/3) requirement

With some support from the system, in the form of certain kind of centralized or distributed reliable and tamperproof service, the number of replicas needed for Byzantine resilience can be reduced from $3f + 1$ to $2f + 1$, which is the minimum possible. Several proposals of this kind have been made since 2004, with the latest [Vero13] possessing performance and efficiency advantages over the earlier ones.

Problems

27.1 Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

27.2 Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

27.3 Weighted threshold voting

Consider a 4-input weighted threshold voting unit with input weights $v + 3$, $v + 2$, $v + 1$, v, and a threshold of $2v + 4$, where v is an integer.

a. Show, with a precise and convincing proof, that the given voting unit is equivalent to one with input weights 3, 2, 2, 1 and threshold 5.
b. What are the practical implications of the equivalence of part a?
c. If we change the threshold to $2v + 3$, what would be a simplified set of weights and threshold?
d. Show a possible circuit realization for the voting unit in part c, using an ordinary nonweighted voting unit.

27.4 Nonuniform voting schemes

The United Nations Security Council consists of five permanent members and 10 nonpermanent members that serve two-year terms. For a resolution to be approved by the Council, all five permanent members and at least four nonpermanent members must agree. Can this decision scheme be formulated as weighted voting? How, or why not?

27.5 Weighted threshold voting

Find a simpler set of weights, and the associated threshold value, that would produce results identical to each of the following weighted threshold voting schemes, or show that no simplification is possible.

a. Weights: 3, 3, 2, 2, 2. Threshold: 8 (Answer: 2, 2, 1, 1, 1; 5)
b. Weights: 5, 4, 3, 2, 1. Threshold: 9 (Solution: Calling the simplified weights a, b, c, d, and e, respectively, the minimal sets are {a, b}, {a, c, d}, {a, c, e}, {b, c, d}. No simplification is possible, as no two votes can be equal to each other.}
c. Weights: $k + 1$, $k + 1$, $k + 1$, $k + 1$, $k + 1$, 1, 1, 1, . . . , 1 (k 1s in all). Threshold: $5k + 9$.

27.6 Generalized and weighted voting

A generalized voting scheme can be specified by listing its agreement sets. For example, simple 2-out-of-3 majority voting with inputs A, B, and C has the agreement sets $\{A, B\}$, $\{B, C\}$, $\{C, A\}$. Show that each of the agreement sets below corresponds to a weighted threshold voting scheme and present a simple hardware voting unit implementation for each case.

a. $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{B, C, D\}$

b. $\{A, B\}$, $\{A, C, D\}$, $\{B, C, D\}$

c. $\{A, B, C\}$, $\{A, C, D\}$, $\{B, C, D\}$

d. $\{A, B\}$, $\{A, C, D\}$, $\{B, D\}$

27.7 Generalized and weighted voting

Is there any generalized voting scheme on 4 inputs that cannot be realized as a weighted threshold voting scheme? Fully justify your answer.

27.8 Assent in social choice

Problem to be designed based on [Bald13].

27.9 Computation of majority

Problem to be designed based on [DeMa12].

References and Further Readings

[Bald13] Baldiga, K. A. and J. R. Green, "Assent-Maximizing Social Choice," *Social Choice and Welfare*, Vol. 40, No. 2, pp. 439-460, February 2013.

[Bend15] Bendahmane, A., M. Essaaidi, A. El Moussaoui, and A. Younes, "The Effectiveness of Reputation-Based Voting for Collusion Tolerance in Large-Scale Grids," *IEEE Trans. Dependable and Secure Computing*, Vol. 12, No. 6, pp. 665-674, November-December 2015.

[Cons13] Consortium for Mathematics and Its Applications, *For All Practical Purposes: Mathematical Literacy in Today's World*, W. H. Freeman, 9th ed., 2013, 912 pp. [Part III, Voting and Social Choice: 9. Social Choice: The Impossible Dream; 10. The Manipulability of Voting Systems; 11. Weighted Voting Systems; 12. Electing the President.]

[Corr05] Correia, M., N. F. Neves, L. C. Lung, and P. Verissimo, "Low Complexity Byzantine-Resilient Consensus," *Distributed Computing*, Vol. 17, No. 3, pp. 237-249, March 2005.

[DeMa12] De Marco, G., E. Kranakis, and G. Wiener, "Computing Majority with Triple Queries," *Theoretical Computer Science*, Vol. 461, pp. 17-26, November 2012.

[Deno19] Denoeux, T., "Decision-Making with Belief Functions: A Review," *Int'l J. Approximate Reasoning*, Vol. 109, pp. 87-110, June 2019.

[Frie07] Friedman, R., A. Mostefaoui, S. Rajsbaum, and M. Raynal, "Asynchronous Agreement and Its Relation with Error-Correcting Codes," *IEEE Trans. Computers*, Vol. 56, No. 7, pp. 865-875, July 2007.

[Kilg06] Kilgour, D. M., S. J. Brams, and R. Sanver, "How to Elect a Representative Committee Using Approval Balloting," in [Sime06], pp. 83-95.

[Laha15] Lahat, D., T. Adali, and C. Jutten, "Multimodal Data Fusion: An Overview of Methods, Challenges, and Prospects," *Proc. IEEE*, Vol. 103, No. 9, pp. 1449-1477, September 2015.

[Levi13] Levitin, Gregory and Kjell Hausken, "Defending Threshold Voting Systems with Identical Voting Units," *IEEE Trans. Reliability*, Vol 62, No. 2, pp. 466-477, June 2013.

[Mesk06] Meskanen, T. and H. Nurmi, "Distance from Consensus: A Theme and Variations," in [Sime06], pp. 117-132.

[Nage06] Nagel, J. H., "A Strategic Problem in Approval Voting," in [Sime06], pp. 133-150.

[Parh91] Parhami, B., "Voting Networks," *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991.

[Saar06] Saari, D. G., "Hidden Mathematical Structures of Voting," in [Sime06], pp. 221-234.

[Sime06] Simeone, B. and F. Pukelsheim (eds.), *Mathematics and Democracy: Recent Advances in Voting Systems and Collective Choice*, Springer, 2006.

[Vemp13] Vempaty, A., L. Tong, and P. Varshney, "Distributed Inference with Byzantine Data: State-of-the-Art Review on Data Falsification Attacks," *IEEE Signal Processing*, Vol. 30, No. 5, pp. 65-75, September 2013.

[Vero13] Veronese, G. S., M. Correia, A. N. Bessani, L. C. Lung, and P/ Verissimo, "Efficient Byzantine Fault-Tolerance," *IEEE Trans. Computers*, Vol. 62, No. 1, pp. 16-30, January 2013.

28 Fail-Safe System Design

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

Douglas Adams

"They that can give up essential liberty to obtain a little temporary safety deserve neither liberty nor safety."

Benjamin Franklin

Topics in This Chapter

We would prefer our computers and computer-based systems to be immune to failures. However, given that complete freedom from failures is impossible to ascertain in all but the simplest systems, the next best thing is to follow a fail-safe design paradigm: to ensure that failures do not create safety hazards for humans and do not lead to the loss of valuable resources, including stored data. The long tradition of the field of safety engineering, as applied to older engineering disciplines, is a useful guide for our discussions. Following this guide, we study some methods relating to fail-safe specifications, design of fail-safe logic circuits, and safety concerns at the system and user levels.

28.1 Fail-Safe System Concepts

This section to be written based on the following slides.

Fail-safe: Produces one of a predetermined set of safe outputs when it fails as a result of "undesirable events" that it cannot tolerate

Fail-safe traffic light: Will remain stuck on red

Fail-safe gas range/furnace pilot flame: Cooling off of the pilot assembly due to the flame going out will shut off the gas intake valve

A fail-safe digital system must have at least two binary output lines, together representing the normal outputs and the safe failure condition

Reason: If we have a single output line, then even if one value (say, 0) is inherently safe, the output stuck at the other value would be unsafe

Two-rail encoding is a possible choice: **0**: 01, **1**: 10, **F**: 00, 11, or both

Totally fail-safe: Only safe erroneous outputs are produced, provided another failure does not occur before detection of the current one

Ultimate fail-safe: Only safe erroneous output is produced, forever

28.2 Principles of Safety Engineering

Safety-critical systems exists outside the domain of computing. Thus, over many years of experience with such systems, a number of principles for desining safe systems have been identified. These principles include [Gold87]:

1. Use barriers and interlocks to constrain access to critical system resources or states
2. Perform critical actions incrementally, rather than in a single step
3. Dynamically modify system goals to avoid or mitigate damages
4. Manage the resources needed to deal with a safety crisis, so that enough will be available in an emergency
5. Exercise all critical functions and safety features regularly to assess and maintain their viability
6. Design the operator interface to provide the information and power needed to deal with exceptions
7. Defend the system against malicious attacks

28.3 Fail-Safe Specifications

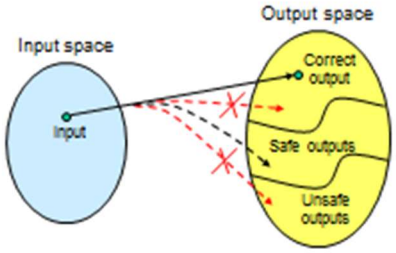
This section to be written based on the following slides.

Example: Amusement park train

Amusement park train safety system

Signal s_B when asserted indicates that the train is at beginning of its track (can move forward, but should not be allowed to go back)

Signal s_E when asserted indicates that the train is at end of its track (can go back, but should not move forward)



Is the specification above consistent and complete?

No, because it does not say what happens if $s_B = s_E = 1$; this would not occur under normal conditions, but because such sensors are often designed to fail in the safe mode, the combination is not impossible

Why is this a problem, though? (Train simply cannot be moved at all)

Completeness will prevent potential implementation or safety problems

Example: Traffic light controller

Today's traffic lights are computer-controlled, with a microprocessor embedded in a control box near the intersection running the requisite software. There are typically many inputs, including data from vehicle sensors embedded in the road, input from pedestrian crossing buttons, and information about traffic conditions received from a remote source or collected from cameras installed at the intersection. The light fixtures themselves may contain conventional green, yellow/amber, and red lights, as well as special turn lights. Finally, there may be various pedestrian signals.

Let us ignore all this complexity and focus on the simplest possible intersection with two sets of 3 lights (green, yellow/amber, red) for the two intersecting roads. The simplest algorithm for controlling the lights would cycle through various states, allowing cars to move on one street for a fixed amount of time and then changing the lights to let cars on the other road to proceed for a while. The state of the intersection at any given time can be represented as XY, where each of the letters X and Y can assume a color code from {G, Y, R}. With regard to safety, the state RR is very safe, but, of course, undesirable from the traffic movement point of view, and the state GG is highly dangerous and

should never occur. One can say that with regard to safety, the states can be ordered in the following way, from the safest to the most dangerous: RR, {RY, YR}, {RG, GR}, YY, {GY, YG}, GG. So, the objective in the design should be to assure that the state of the traffic lights is one of the first five just listed, avowing any of the final four states.

The cycling should be
RG → RY → RR → GR → YR → RR → RG
The RR states, maintained for a few seconds, are inserted to provide a safe buffer betweeh cars moving on the two streets.

Now, if the lights are controlled by six separate signals that turn them on and off, there is a chance that through some signal being stuck or incorrectly computed, one of the prohibited states is entered.

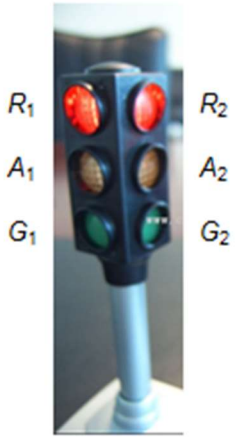
Six signals: red, amber, and green light control for each direction (no left turn signals)

Safety flip-flop, when set, forces flashing red lights in both directions

$S = (A_1 \vee G_1) \wedge (A_2 \vee G_2)$

Let g_1 be the computed green light signal for direction 1, and so on for other signals, and G_1 be the signal that controls the light

$G_1 = \overline{S} \wedge g_1 \wedge \overline{(A_2 \vee G_2)}$



28.4 Fail-Safe Combinational Logic

This section to be written based on the following slides.

Similar to the design of self-checking circuits:
Design units so that for each fault type of interest, the output is either correct or safe

Totally fail-safe: Fail-safe and self-testing [Nico98]

Strongly fail-safe: Intermediate between fail-safe and totally fail-safe
For each fault x it is either totally fail-safe *or* it is fail-safe with respect to x and strongly fail-safe with respect to $x \cup y$, where y is another fault

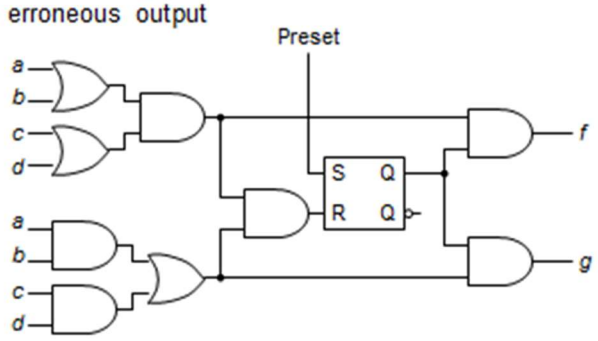
Example: Fail-safe 2-out-of-4 code checker

Input: 4 bits *abcd*, exactly 2 of which must be 1s

Output: *fg* = 01 or 10, if the input is valid
00 safe erroneous output
11 unsafe erroneous output

Codewords

a	b	c	d
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0



Output will become permanently 00 upon the first unsafe condition

28.5 Fail-Safe State Machines

This section to be written based on the following slides.

Use an error code to encode states

Implement the next-state logic so that the machine is forced to an error state when something goes wrong

Possible design methodology:

Use Berger code for states, avoiding the all 0s state with all-1s check, and vice versa

Implement next-state logic equations in sum-of-products form for the main state bits and in product-of-sums form for the check state bits

The resulting state machine will be fail-safe under unidirectional errors

State	Input x=0	Input x=1
A	E	B
B	C	D
C	A	D
D	E	D
E	A	D

State	Encoding
A	001 10
B	010 10
C	011 01
D	100 10
E	101 01

Hardware overhead for n-state machine consists of O(log log n) additional state bits and associated next-state logic, and a Berger code checker connected to state FFs

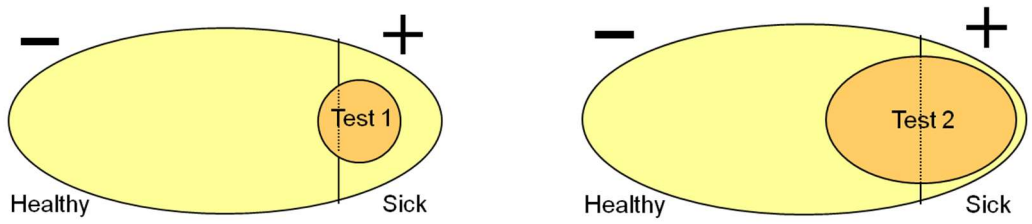
28.6 System- and User-Level Safety

Any fail-safe system will have some false alarms: situations in which the system enters its safe mode of operation, or stops functioning altogether, in the absence of any imminent danger. Conversely, monitoring components in a fail-safe system may miss hazardous conditions, thus allowing the system to remain operational in an unsafe mode. False alarms lead to inconvenience, lower productivity, and perhaps financial losses. However, these undesirable consequences are preferable to potentially huge losses that might result from unsafe operation. This tradeoff between the frequency of false alarms and the likelihood of missed warnings is a fundamental one in any system that relies on tests with binary outcomes for diagnostics.

A binary test is characterized by its *sensitivity* and *specificity*. Sensitivity is the fraction of positive cases (e.g., people having a particular disease) that the test diagnoses correctly. Specificity is the fraction of negative cases (e.g., healthy people) that are correctly identified. Figure 28.test contains a graphical depiction of the notions of test specificity and sensitivity, as well as the trade-offs between them. We see that Test 1 in Fig. 28.xa is high on specificity, because it does not lead to very many false positives, whereas Test 2 in Fig. 28.xb is highly sensitive in the sense of correctly diagnosing a great majority of the positive cases. In general, highly sensitive tests tend to be less specific, and vice versa. For safety-critical systems, we want to err on the side of too much sensitivity.

A *dead-man's switch*, sometimes called a *kill switch*, is used in a number of safety-critical systems that should not be operable without an operator's presence. It usually, takes the form of a handle or pedal that the operator must touch or press continuously, and was first used in trains to stop them in the event of operator incapacitation. Early trains serving urban areas carried two operators, so that the second one could take over if the first operator was incapacitated for any reason, much like modern passenger aircraft using a pilot and a co-pilot. The invention of dead man's switch came about as a cost-saving measure.

Within a self-monitoring safety-critical system, a dead man's switch can take the form of a monitoring unit that continuously runs tests to verify the safety of the current configuration and operational status. If some predetermined time interval passes without the monitor asserting itself or confirming proper status, the system will automatically enter its safe operating mode.



(a) Specific test (b) Sensitive test

Fig. 28.test The notions of specificity and sensitivity in binary tests.

Interlocks, watchdog units, and other preventive and monitoring mechanisms are commonplace in safety-critical systems.

Problems

28.1 Exception-free arithmetic

One of the sources of difficulty in designing dependable systems is dealing with arithmetic exceptions, such as overflow, underflow, and various disallowed or illegal operations. Read the paper [Haye09] and answer the following questions.

a. What is exception-free arithmetic and how can it be implemented?

b. What is the performance overhead, if any, of exception-free arithmetic?

c. How does exception-freedom contribute to dependability?

d. Does the method proposed in the paper eliminate all exceptions or only some of them?

References and Further Readings

[Chua78] Chuang, H. and S. Das, "Design of Fail-Safe Sequential Machines Using Separable Codes," *IEEE Trans. Computers*, Vol. 27, No. 3, pp. 249-251, March 1978.

[Haye09] Hayes, B., "The Higher Arithmetic: How to Count to a Zillion without Falling off the End of the Number Line," *American Scientist*, Vol. 97, No. 5, pp. 364-368, September-October 2009.

[Nico98] Nicolaidis, M., "Fail-Safe Interfaces for VLSI: Theoretical Foundations and Implementation," *IEEE Trans. Computers*, Vol. 47, No. 1, pp. 62-77, January 1998.

A Past, Present, and Future

"You can't change the past, but you can ruin the present by worrying about the future."

Anonymous

"The distinction between the past, present and future is only a stubbornly persistent illusion."

Albert Einstein

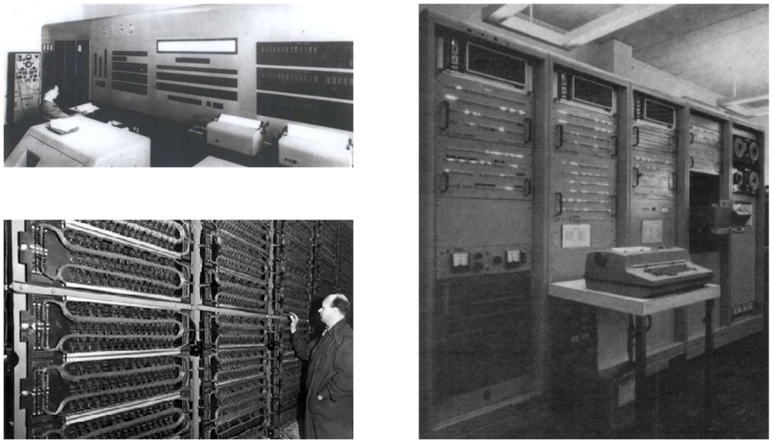
Topics in This Appendix

In this appendix, we trace the history of dependable computing, from the earliest digital computers to modern machines used in a variety of application domains, from space exploration and real-time process control to banking and e-commerce. We explore a few turning points along this impressive historical path, including emergence of long-life systems for environments that make repair impossible, development of safety-centered computer systems, meeting the stringent demands of applications requiring high availability, and interaction of dependability with security concerns. After discussing how advanced ideas and methods developed in connection with system classes named above find their way into run-of-the-mill commercial and personal systems, we conclude with a discussion of current trends, future outlook, and resources for further study of dependable computing.

A.1 Historical Perspective

Even though fault-tolerant computing as a discipline had its origins in the late 1960s, key activities in the field began some two decades earlier in Prague, where computer scientist Antonin Svoboda (1907-1980) built the SAPO computer. The pioneering design of SAPO employed triplication and voting to overcome the effects of poor component quality. Svoboda's efforts were little known in the West and thus did not exert much influence on the field as we know it. JPL's STAR (self-testing-and-repairing) computer, on the other hand, was highly influential, both owing to its design innovations and because the project leader, Professor Algirdas Avizienis, was one of the early leaders of the field in the US. The STAR computer project was motivated by NASA's plans for a Solar System exploration mission taking 10 years. Known as "The Grand Tour," the mission was later scrapped in its original form, but a number of its pieces, including the highly fault-tolerant computer, formed the basis of later space endeavors.

With this introduction, we embark on reviewing the development of dependable and fault-tolerant computing by discussing each of the seven decades 1950-2010 very briefly in the rest of this section.



(a) SAPO; Prague, 1951 ` (b) STAR; JPL, 1971

Fig. A.early Two early fault-tolerant computers.

Dependable computing in the 1950s

SAPO, built in Czechoslovakia in 1951 by Antonin Svoboda
Used magnetic drums and relays

SAPO fault tolerance features were motivated by [Renn84]:
Low quality of components available to designers
Severe (political) pressure to get things right

With the advent of transistors, which were much more reliable than relays and vacuum tubes, redundancy methods took a back seat (used only for exotic or highly critical systems)

Dependable computing in the 1960s

NASA began extensive development in long-life, no-maintenance computer systems for (manned) space flight

Orbiting Astronomical Observatory (OAO), transistor-level fault masking

The Apollo Guidance System, triplication and voting

Mars Voyager Program (late 1960s, cancelled)

Deep space missions (e.g., the cancelled Solar System Grand Tour)
JPL self-testing-and-repairing computer (STAR)

Also:
AT&T Electronic Switching System (ESS), down time of minutes/decade
Serviceability features in mainframes, such as IBM System/360

Dependable computing in the 1970s

Two influential avionics systems:
Software-implemented fault tolerance (SIFT), SRI, Stanford Univ.
  Spun off a commercial venture, August Systems (process control)
Fault-tolerant multiprocessor (FTMP), Draper Lab., MIT

The Space Shuttle (4 identical computers, and a 5th one running different software developed by a second contractor)

Redundancy in control computers for transportation (trains)

Also:
First Fault-Tolerant Computing Symposium, FTCS (1971)
Systems for nuclear reactor safety
Tandem NonStop (1976), transaction processing

Dependable computing in the 1980s

1980: IFIP WG 10.4 on Dependable Computing and Fault Tolerance

Continued growth in commercial multiprocessors (Tandem)

Advanced avionics systems (Airbus)

Automotive electronics

Reliable distributed systems (banking, transaction processing)

Dependable computing in the 1990s

DCCA conferences, started in 1989, continued until 1999 (DCCA-7); subsequently merged with FTCS to become DSN Conf.

SEU tolerance

BIST

IBM G4 fault-tolerant mainframe

Robust communication and collaboration protocols (routing, broadcast, f-t CORBA)

Dependable computing in the 2000s

2000: First DSN Conf. (DSN-2000, 30th in series going back to FTCS)
Now composed of two symposia
DCCS: Dependable Computing and Communications Symp.
PDS: Performance and Dependability Symp.

2004: *IEEE Trans. Dependable and Secure Computing*

Deep submicron effects

F-T ARM, Sparc

Intrusion detection and defense

Widespread use of server farms for directory sites and e-commerce (Google, e-Bay, Amazon)

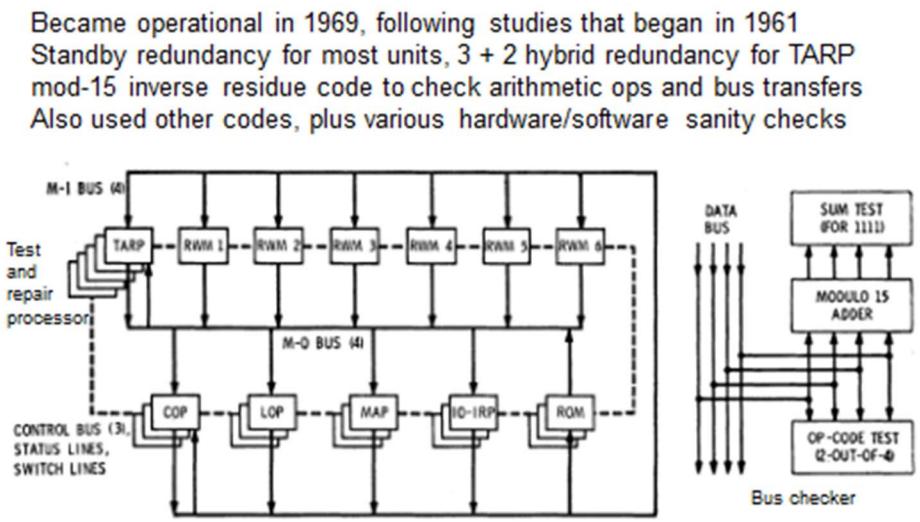
In the 2010s decade, we have continued to see the development and further maturation of the field. New challenges to be faced in the rest of this decade include applying and extending available techniques to new computing paradigms and environments, auch as cloud computing and its attendant mobile platforms (smartphones and compact tablets). Among problems that need extensive study and innovative solutions is greater integration of reliability and security concerns. At the end of this decade, the field of dependable computing, and its flagship conference, will be preparing to celebrate their 50th anniversary (DSN-50 will be held in 2020): a milestone that should be cause for a detailed retrospective assessment of, and prospective planning for, the methods and traditions of the field, as they have been practiced and as they might apply to new and emerging technologies. By then, some of the pioneers of the field will no longer be with us, but there are ample new researchers to carry the torch forward.

A.2 Long-Life Systems

Interest in long-life computer systems began with the desire to launch spacecraft on multiyear missions, where there is no possibility of repair. Even for manned space missions, the limited repair capacity isn't enough to offset the effects of even greater reliability requirements. Today, we are still interested in long-life system for space travel, but the need for such systems has expanded owing to many remotely located or hard-to-access systems for intelligence gathering and environmental monitoring, to name only two application domains.

Systems of historical and current interest that fall into the long-life category include NASA's OAO, the Galileo spacecraft, JPL STAR, the International Space Station, communication satellites, and remote sensor networks.

Description of the JPL STAR computer

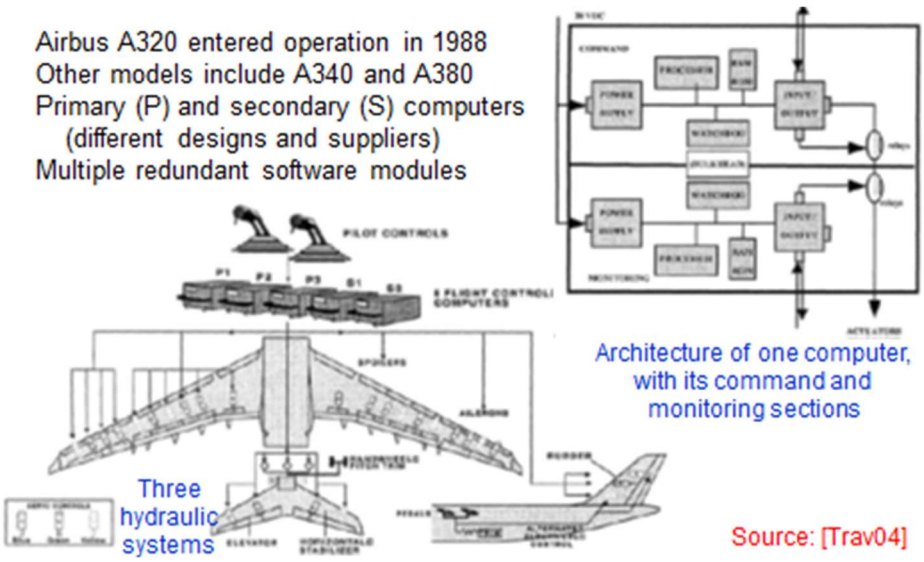


A.3 Safety-Critical Systems

Safety-critical computer systems were first employed in flight control, nuclear reactor safety, and factory automation. Today, the scope of safety-critical systems has broadened substantially, given the expansion of numerous applications requiring computerized control: high-speed transportation, health monitoring, surgical robots.

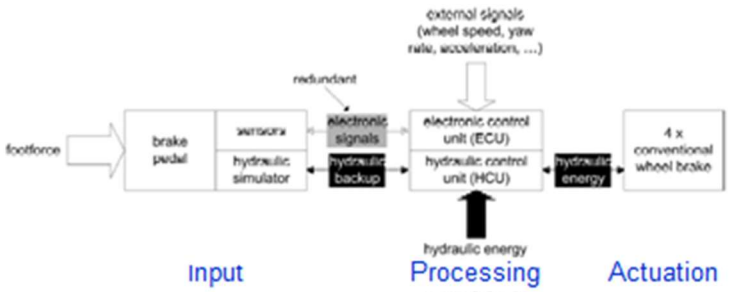
Systems of historical and current interest that fall into the safety-critical category include Carnegie Mellon University's C.vmp, Stanford University's SIFT, MIT's FTMP, the industrial control computers of August Systems, high-speed train controls, and automotive computers

Avionics fly-by-wire systems

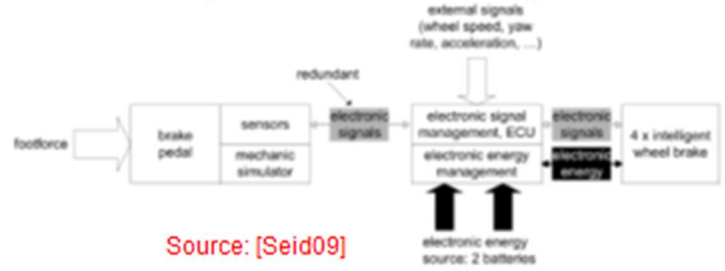


Automotive drive-by-wire systems

Interim braking solution with mixed electronics and hydraulics



Fully electronic, with redundant buses, and power supplies



Source: [Seid09]

A.4 High-Availability Systems

High-availability computer systems were first developed by telephone companies for their electronic switching systems. Even short interruptions in telephone service are both embarrassing to communication companies and sources of significant revenue loss. Today, the availability of communication services is even more important, not only for telephony but also for a host of applications such as on-line banking, e-commerce, social networking, and other systems that can ill-afford even very short down times.

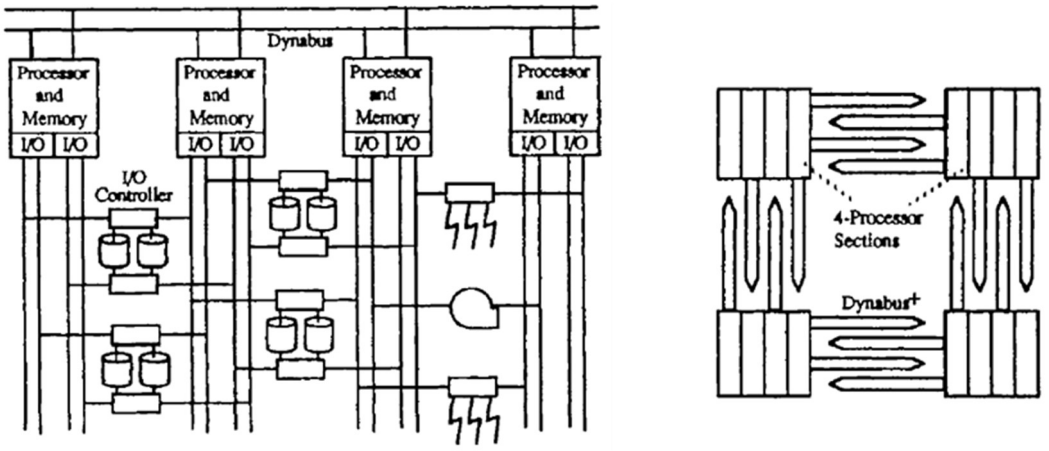
Systems of historical and current interest that fall into the high-availability category include AT&T ESS 1-5 (telephone switching, 1965-1982), Tandem's various computers (NonStop I-II, . . . , Cyclone, CLX800, 1976-1991), Stratus FT200-XA2000 (1981-1990), banking systems, Internet portals (Google, Yahoo), and e-commerce (Amazon, eBay).

.

The following description of Tandem NonStop Cyclone is copied from [Parh99].

The first Tandem NonStop, a MIMD distributed-memory bus-based multiprocessor, was announced in 1976 for database and transaction processing applications requiring high reliability and data integrity. Since then, several versions have appeared. The Tandem NonStop Cyclone, described in this section [Prad96], was first introduced in 1989. A main objective of Cyclone's design was to prevent any single hardware or software malfunction from disabling the system. This objective was achieved by hardware and informational redundancy as well as procedural safeguards, backup processes, consistency checks, and recovery schemes.

A fully configured cyclone system consisted of 16 processors that were organized into sections of 4 processors. Processors in each section were interconnected by a pair of 20 MB/s buses (Dynabus) and each could support two I/O subsystems capable of burst transfer rates of 5 MB/s (Fig. A.Tand-a). An I/O subsystem consisted of two I/O channels, each supporting up to 32 I/O controllers. Multiple independent paths were provided to each I/O device via redundant I/O subsystems, channels, and controllers. Up to 4 sections could be linked via unidirectional fiber optics Dynabus$^+$ that allowed multiple sections to be nonadjacent within a room or even housed in separate rooms (Fig. A.Tand-b). By Isolating Dynabus$^+$ from Dynabus, full-bandwidth communications could occur independently in each 4-processor section. Other features of the NonStop Cyclone are briefly reviewed below.



(a) One section of NonStop Cyclone system (b) Dynabus+ connecting 4 sections

Fig. A.Tand Tandem NonStop Cyclone system of the late 1980s.

Processors: Cyclone's 32-bit processors had advanced superscalar CISC designs. They used dual 8-stage pipelines, an instruction pairing technique for instruction-level parallel processing, sophisticated branch predication algorithms for minimizing pipeline bubbles, and separate 64 KB instruction and data caches. Up to 128 MB of main memory could be provided for each cyclone processor. The main memory was protected against errors through the application of a SEC/DED code. Data transfers between memory and I/O channels were performed via DMA and thus did not interfere with continued instruction processing.

System performance: Performance estimates published in 1990 indicated that, after accounting for cache misses and other overhead, the custom-designed Cyclone processor could execute each instruction in an average of 1.5-2 clock cycles. Thus, with a clock period of 10 ns, the peak performance of a fully configured NonStop Cyclone system was about 1000 MIPS. Since each of the two I/O subsystems connected to a processor could transmit data at a burst rate of 5 MB/s, a peak aggregate I/O bandwidth of 160 MB/s was available.

Hardware reliability: Use of multiple processors, buses, power supplies, I/O paths, and mirrored disks were among the methods used to ensure continuous operation despite hardware malfunctions. A fail-fast strategy was employed to reduce the possibility of error propagation and data contamination. Packaging and cooling technologies had also been selected to minimize the probability of failure and to allow components, such as

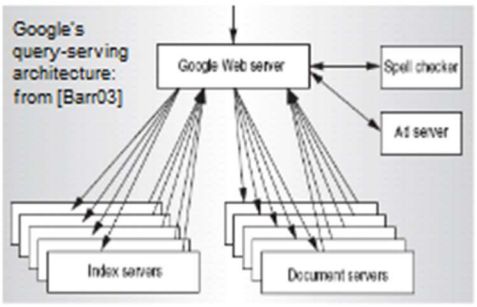
circuit boards, fans, and power supplies, to be hot-pluggable without a need to interrupt system operation. When a malfunctioning processor was detected via built-in hardware self-checking logic, its load was transparently distributed to other processors by the operating system.

Software reliability: The GUARDIAN 90 operating system was a key to Cyclone's high performance and reliability. Every second, each processor was required to send an "I'm alive" message to every other processor over all buses. Every 2 seconds, each processor checked to see if it had received a message from every other processor. If a message had not been received from a particular processor, it was assumed to be malfunctioning. Other software mechanisms for malfunction detection included data consistency checks and kernel-level assertions. Malfunctions in buses, I/O paths, and memory were tolerated by avoiding the malfunctioning unit or path. Processor malfunctions led to deactivation of the processor. For critical applications, GUARDIAN 90 maintained duplicate backup processes on disjoint processors. To reduce overhead, the backup process was normally inactive but was kept consistent with the primary process via periodic checkpointing messages. Upon malfunction detection, the backup process was started from the last checkpoint, perhaps using mirror copies of the data.

Related systems: In addition to NonStop Cyclone, and its subsequent RISC-based Himalaya servers, Tandem offered the Unix-based Integrity S2 uniprocessor system which tolerated malfunctions via triplication and voting. It used R4000 RISC processors and offered both application and hardware-level compatibility with other Unix-based systems. Commercial reliable multiprocessors were also offered by Stratus (XA/R Series 300, using a pair-and-spare hardware redundancy scheme) and Sequoia (Series 400, using self-checking modules with duplication). Both of the latter systems were tightly coupled shared-memory multiprocessors.

Example: Google data center architecture

Uses commodity (COTS) processing and communication resources
Data replication and software used to handle reliability issues
Massive hardware replication, needed for capacity and high performance, also leads to fault tolerance
Consistency problems are simplified by forcing read-only operation most of the time and shutting down sections of the system for infrequent updates

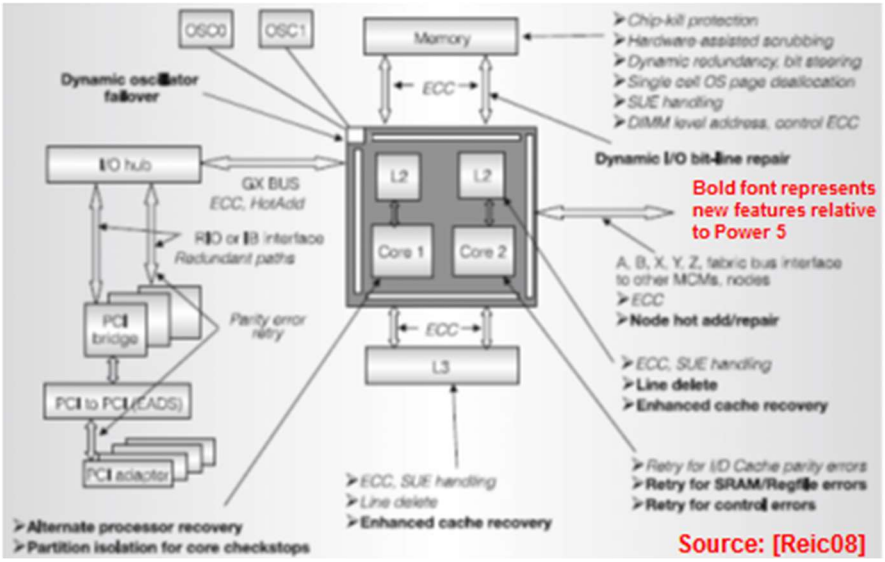


A.5 Commercial and Personal Systems

Due to highly unreliable components, early computers had extensive provisions for fault masking and error detection/correction. Today's components are ultrareliable, but there are so many of them in a typical system that faults/errors/malfunctions are inevitable. It is also the case that computers are being used not only by those with hardware/software training but predominantly by nonexperts, or experts in other fields, who would be immensely inconvenienced by service interruptions and erroneous results.

Systems of historical and current interest that fall into the commercial/personal category include SAPO, IBM System 360, and IBM Power 6 [Reic08].

Description of the IBM Power 6 microprocessor



A.6 Trends, Outlook, and Resources

In this section, we review some of the active research areas in dependable computing and provide a forecast of where the field is headed in the next few decades. Technology forecasting is, of course, a perilous task and many forecasters look quite silly when their predications are examined decades hence. Examples of off-the-mark forecasts include Thomas J. Watson's "I think there is a world market for maybe five computers," and Ken Olson's "There is no reason anyone would want a computer in their home." Despite these and other spectacular failures in forecasting, and cautionary anonymous bits of advice such as "There are two kinds of forecasts: lucky and wrong," I am going to heed the more optimistic musing of Henri Poincare, who said "It is far better to foresee even without certainty than not to foresee at all."

Dependable computer systems and design methods continue to evolve. Over the years, emphasis in the field has shifted from building limited-edition systems with custom components to using commercial off-the-shelf (COTS) components to the extent possible. This strategy implies incorporating dependability features through layers of software and services that run on otherwise untrustworthy computing elements. Designers of COTS components in turn provide features that enable and facilitate such a layered approach. This trend, combined with changes in technology and scale of systems, creates a number of challenges which will occupy computer system designers for decades to come.

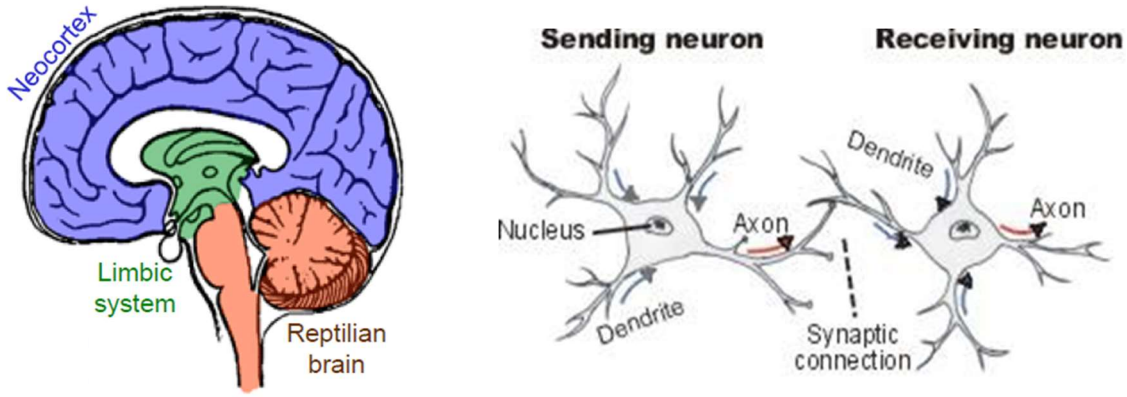
Challenge 1: The curse of shrinking electronic devices (nanotechnology)
Challenge 2: Reliability in cloud computing (opportunities and problems)
Challenge 3: Smarter and lower-power redundancy schemes (brain-inspired)
Challenge 4: Ensuring data longevity over centuries, even millennia
Challenge 5: Dependability verification through reasoning about uncertainty
Challenge 6: Counteracting the combined effects of faults and intrusions
Challenge 7: Reliability as a roadblock for exascale systems (reliability wall)

Nanotechnology brings about smaller and more energy-efficient devices, but it also creates new challenges. Smaller devices are prone to unreliable operation, present complex modeling problems (due to the need for taking parameter variations into account), and exacerbate the already difficult testing problems.

Parameter variations [Ghos10] occur from one wafer to another, between dies on the same wafer, and within each die. Reasons for parameter variations include vertical nonuniformities in the layer thicknesses and horizontal nonuniformities in line and spacing widths.

Cloud computing has been presented as a solution to all corporate and personal computing problems, but we must be aware of its complicated reliability and availability problems [Baue12]. Whereas it is true that multiplicity and diversity of resources can lead to higher reliability by avoiding single points of failure, this benefit does not come about automatically. Accidental and deliberate outages are real possibilities and identifying the weakest link in this regard is no easy task. Assignment of blame in the event of failures and rigorous risk assessment for both e-commerce and safety-critical systems are among other difficulties.

The human brain is often viewed as an epitome of design elegance and efficiency. Though quite energy-efficient (the brain uses around 20 W of power, whereas performing a minute fraction of the brain's function for a short period of time via simulation requires supercomputers that consume megawatts of popwer), its design is anything but elegant. The brain has grown in layers over millions of years of evolutionary time. Newer capabilities are housed in the neocortex, developed fairly recently, and the older reptilian brain parts are still there (Fig. A.brain-a). As a result, there is functional redundancy, meaning that the same function (such as vision) may be performed in multiple regions. Futhermore, the use of a fairly small number of building blocks make it easy for one part of the brain to cover for other parts when they are damaged. The death or disconnection of a small number of neurons is often inconsequential and those suffering substantial brain injuries can and do recover to full brian function. The brain uses electrochemical signaling that is extremely slow compared with electronic communication in modern digital computers. Yet, the overall computational power of the brain is as yet unmatched by even the most powerful supercomputers. Finally, memory and computational functionalities are intermixed in the brain: there aren't a separate memory units and computational elements.



(a) Main parts of the human brain (b) Communication between neurons

Fig. A.brain The human brain's redundancy and uniformity of parts make it more reliable.

As for data longevity, typical media used for mass data storage have lifespans of 3-20 years. Data can be lost to both media decay and format obsolescence. [Elaborate]

Data preservation is particularly daunting when documents and records are produced and consumed in digital form, as such data files have no nondigital back-up.

The field of dependable computing must deal with uncertainties of various kinds. Therefore, research in this field has entailed methods for representing and reasoning about uncertainties. Uncertainties can exist in both data (missing, imprecise, or estimated data) and in rules for processing data (e.g., empirical rules). Approches used to date fall under four categories:

Probabilistic analysis (Bayesian logic)
Certainty/Confidence factors
Evidence/Belief theory (Dempster-Shafer theory)
Continuous truth values (Fuzzy logic)

The interaction between failures and intrusions (security breaches) has become quite important in recent year. Increasingly, hackers on computer systems take advantage of extended system vulnerabilities during failure episodes to compromise such systems. [Elaborate]

As top-of-the-line supercomputers use more and more processing nodes (currently in the hundreds of thousands, soon to be in the millions), system MTBF is shortened and the reliability overhead increases. Checkpointing, for example, must be done more frequently, which can lead to superlinear overhead of such methods in terms of the number p of processors. Typically, computational speed-up increases with p (albeit sometimes facing a flattening due to Amdahl's law). The existence of reliability overhead may mean that the speed-up can actually decline beyond a certain number of processors.

[See Problem 3.17 in Chapter 3 on modeling]
Amdahl's speedup formula $p/[1 + f(p - 1)]$; cannot exceed $1/f$, no matter how large f; tends to a constant as p approaches infinity
Gustafson's constant-running-time scaled speedup $f + p(1 - f)$; nearly p for small f; tends to infinity as p approaches infinity
If the reliability overhead is superlinear in p, then a reliability wall exists
Theorem: Reliability wall exists iff as p approaches infinity the speedup remains finite

Other topics of interest:
Linear threshold gates [Ayme12]
Verifying computations without reexecuting them [Walf15]

The field of dependable computing has undergone many changes since the appearance of the first computer incorporating error detection and correction [Svob79]. The emergence of new technologies and the unwavering quest for higher performance, usability, and reliability are bound to create new challenges in the coming years. These will include completely new challenges, as well as novel or transformed versions of the ones discussed in the preceding paragraphs. Researchers in dependable computing, who helped make digital computers into indispensable tools in the six-plus decades since the introduction of fault tolerance notions, will thus have a significant role to play in making them even more reliable and ubiquitous as we proceed through the second decade of the Twenty-First Century.

Fault tolerance strategies to date have by and large relied on perfect hardware and/or perfect detection of failures. Thus, we either get the correct results (nearly all of the time) or hit an exceptional failure condition, which we detect and recover from. With modern computing technology, where extreme complexity and manufacturing variability make failures the norm, rather than the exception, we should design computers more like biological systems in which failures (and noise) are routinely expected.

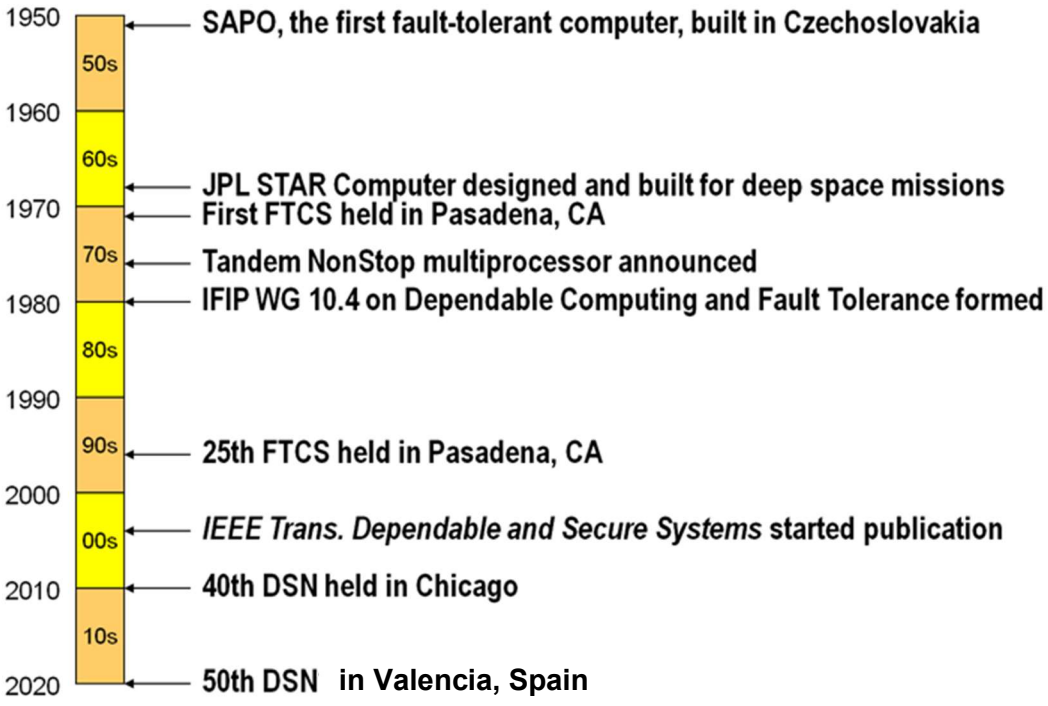


Fig. A.tmIn Dependable computing through the decades. [Incomplete]

We summarize our discussions of the history, current status, and future of dependable computing in the timeline depicted in Fig. A.tmln. As for resources that would allow the reader to gain additional insights in dependable computing, we have already listed some general references at the end of the Preface and topic-specific references at the end of each chapter (and this appendix). Other resources, which are nowadays quite extensive, thanks to electronic information dissemination, can be found through Internet search engines. For example, a search for "fault-tolerant computer" on Google yields some 1.5 million items, not counting additional hits for "dependable computing," "computer system reliability," and other related terms. The author maintains a list of Web resources for dependable computing on his companion website for this book: it can be reached via the author's faculty Web site at University of California, Santa Barbara (UCSB).

Resources for dependable computing

IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance
IFIP WG 10.4 — http://www.dependability.org/wg10.4/

IEEE/IFIP Int'l Conf. Dependable Syst's and Networks (2013, Budapest)
DSN — http://www.dsn.org/

European Dependable Computing Conf. (2012, Sibiu, Romania)
EDCC — http://edcc.dependability.org/

IEEE Pacific Rim Int'l Symp. Dependable Computing (2013, Vancouver)
PRDC — http://www.ksl.ci.kyutech.ac.jp/prdc2010/

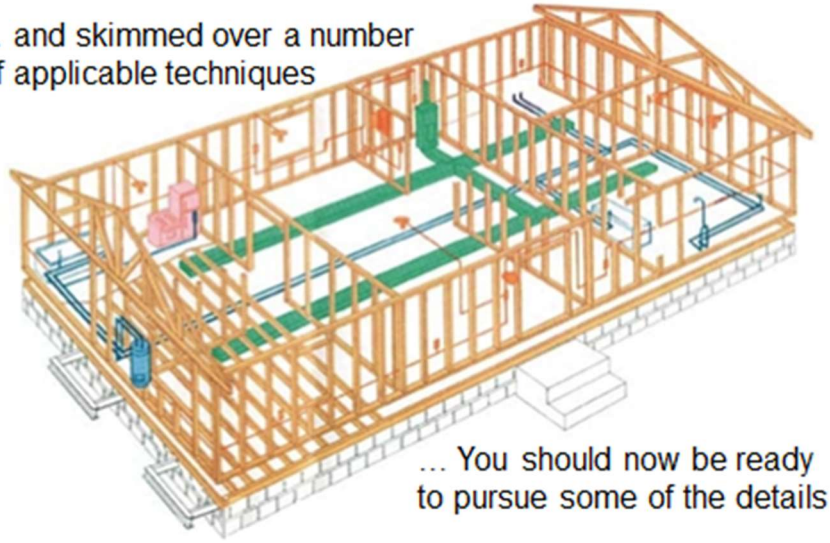
Int'l Conf. Computer Safety, Reliability and Security (2013, Toulouse)
SAFECOMP — http://www.safecomp.org/

IEEE Trans. Dependable and Secure Computing (since 2004)
IEEE TDSC — http://www.computer.org/portal/web/tdsc/

In this book, we have built a framework and skimmed over a number of applicable techniques at various levels of the system hierarchy, from devices to applications. This framework can serve as your guide, allowing you to integrate new knowledge you gain in the field and to see how it fits in the dependable computing universe.

We Have Built a Framework

... and skimmed over a number of applicable techniques



... You should now be ready to pursue some of the details

Problems

A.1 Safety-critical systems

It was reported widely that on November 19, 2009, a computer failure at the US Federal Aviation Administration (FAA) caused massive flight cancellations and delays. The failure made some flight data (such as flight numbers, destinations, and altitudes) unavailable, leading to manual data entry and forcing flight controllers to space the aircraft further apart for safety reasons. Initially, very few technical details were available about the incident, with news reports blaming it on the failure of a single circuit board in Salt Lake City, Utah. However, more details have emerged since then. Prepare a 2-page report about this incident, citing the main technical reasons for the disruption and the role played by dependability enhancement features, or lack thereof.

A.2 Data privacy and government access privileges

There have been suggestions in recent years to make encrypted private communications accessible to certain government entities. The article [Neum15] likens such privacy circumvention methods to placing house keys under doormats, with the implication that backdoor access channels can be abused. Prepare a half-page, single-spaced abstract for the article in which you cite a couple of key reasons for and against such mandated access.

A.3 Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

References and Further Readings

[Aviz71] Avizienis, A., G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, and D. K. Rubin, "The STAR (Self-Testing-And-Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Trans. Computers*, Vol. 20, No. 11, pp. 1312-1321, November 1971.

[Aviz87] Avizienis, A., H. Kopetz, and J.-C. Laprie (eds.), *The Evolution of Fault-Tolerant Computing* (Dependable Computing and Fault-Tolerant Systems, Vol. 1), Springer, 1987.

[Ayme12] Aymerich, N. and A. Rubio, "Fault-Tolerant Nanoscale Architecture Based on Linear Threshold Gates," *Microprocessors and Microsystems*, Vol. 36, No. 5, pp. 420-426, July 2012.

[Barr03] Barroso, L. A., J. Dean, abd U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, Vol. 23, No. 2, pp. 22-28, March-April 2003.

[Bart07] Bartley, G. F., "Boeing B-777: Fly-By-Wire Flight Controls," in *Digital Avionics Handbook*, ed. by C. R. Spitzer, Vol. 1, pp. 23-1 to 23-14, CRC Press, 2006. Electronic version of the handbook available at: http://www.engnetbase.com/books/5549/8438fm.pdf

[Baue12] Bauer, E. and R. Adams, *Reliability and Availability of Cloud Computing*, IEEE Press, 2012.

[Cart64] Carter, W. C., H. C. Montgomery, R. J. Preiss, and H. J. Reinheimer, "Design of Serviceability Features for the IBM System/360," *IBM J. Research and Development*, Vol. 8, No. 2, pp. 115-126, April 1964.

[Ghos10] Ghosh, S. and K. Roy, "Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale era," *Proceedings of the IEEE*, Vol. 98, No. 10, pp. 1718-1751. October 2010.

[Han05] Han, J., E. Taylor, J. Gao, and J. Fortes, "Reliability Modeling of Nanoelectronic Cirsuits," *Proc. 5th IEEE Conf. Nanotechnology*, Vol. 1, pp. 104-107, July 2005.

[Hitt07] Hitt, E. F. and D. Mulcare, "Fault-Tolerant Avionics," in *Digital Avionics Handbook*, ed. by C. R. Spitzer, Vol. 2, pp. 8-1 to 8-24, CRC Press, 2007. Electronic version of the handbook available at: http://www.engnetbase.com/books/6327/8441_fm.pdf

[Lati18] Latif, S. S., "Fault Tolerance in Reversible Logic," MS thesis, U. Lethbridge, 2018.

[Lin14] Lin, C.-C., A. Chakrabarti, and N. K. Jha, "FTQLS: Fault-Tolerant Quantum Logic Synthesis," *IEEE Trans. VLSI Systems*, Vol. 22, No. 6, pp. 1350-1363, June 2014.

[Mack07] Mack, M. J., W. M. Sauer, S. B. Swaney, and B. G. Mealey, "IBM Power6 Reliability," *IBM J. Research and Development*, Vol. 51, No. 6, pp. 763-, 2007.

[Msad15] Msadek, N., R. Kiefhaber, and T. Ungerer, "A Trustworthy, Fault-Tolerant and Scalable Self-Configuration Algorithm for Organic Computing Systems," *J. System Architecture*, Vol. 61, No. 10, pp. 511-519, November 2015.

[Neum15] Neumann, P. G. et al., "Inside Risks: Keys under Doormats," *Communications of the ACM*, Vol. 58, No. 10, pp. 24-26, October 2015.

[Parh99] Parhami, B., *Introduction to Parallel Processing: Algorithms and Architectures*, Plenum, 1999.

[Prad96] Pradhan, D. K., "Case Studies in Fault-Tolerant Multiprocessor and Distributed Systems," Chapter 4 in *Fault-Tolerant Computer System Design,* Prentice–Hall, 1996, pp. 236–281.

[Reic08] Reick, K., P. N. Sanda, S. Swaney, J. W. Kellington, M. Mack, M. Floyd, and D. Henderson, "Fault-Tolerant Design of the IBM Power6 Microprocessor," *IEEE Micro*, Vol. 28, No. 2, pp. 30-38, March-April 2008.

[Renn84] Rennels, D. A., "Fault-Tolerant Computing—Concepts and Examples," *IEEE Trans. Computers*, Vol. 33, No. 12, pp. 1116-1129, December 1984.

[Seid09] Seidel, F., "X-by-Wire," accessed online on December 2, 2009, at: http://osg.informatik.tu-chemnitz.de/lehre/old/ws0809/sem/online/x-by-wire.pdf

[Siew95] Siewiorek, D. (ed.), *Fault-Tolerant Computing Highlights from 25 Years*, Special Volume of the 25th Int'l Symp. Fault-Tolerant Computing, 1995.

[Sing18] Singh, G., B. Raj, and R. K. Sarin, "Fault-Tolerant Design and Analysis of QCA-Based Circuits," *IET Circuits, Devices & Systems*, Vol. 12, No. 5, pp. 638-644, 2018.

[Svob79] Svoboda. A., "Oral History Interview OH35 by R. Mapstone," 15 November 1979, Charles Babbage Institute, University of Minnesota, Minneapolis. Transcripts available at: http://www.cbi.umn.edu/oh/pdf.phtml?id=263

[Trav04] Traverse, P., I. Lacaze, and J. Souyris, "Airbus Fly-by-Wire: A Total Approach to Dependability," in *Building the Information Society*, ed. by P. Jacquart, pp. 191-212, Springer, 2004.

[Vinh16] Vinh, P. C. and E. Vassev, "Nature-Inspired Computation and Communication: A Formal Approach," *Future Generation Computer Systems*, Vol. 56, pp. 121-123, March 2016.

[Walf15] Walfish, M. and A. J. Blumberg, "Verifying Computations without Reexecuting Them," *Communications of the ACM*, Vol. 58, No. 2, pp. 74-84, February 2015.

[Yang12] Yang, X., Z. Wang, J. Xue, and Y. Zhou, "The Reliability Wall for Exascale Supercomputing," *IEEE Trans. Computers*, Vol. 61, No. 6, pp. 767-779, June 2012.