Fault tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of one or more faults within some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system, in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability, mission-critical, or even life-critical systems. The ability of maintaining functionality when portions of a system break down is referred to as **graceful degradation**.[1]

A **fault-tolerant design** enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.[2] The term is most commonly used to describe computer systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software. An example in another field is a motor vehicle designed so it will continue to be drivable if one of the tires is punctured, or a structure that is able to retain its integrity in the presence of damage due to causes such as fatigue, corrosion, manufacturing flaws, or impact.

Within the scope of an *individual* system, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them, and, in general, aiming for self-stabilization so that the system converges towards an error-free state. However, if the consequences of a system failure are catastrophic, or the cost of making it sufficiently reliable is very high, a better solution may be to use some form of duplication. In any case, if the

consequence of a system failure is so catastrophic, the system must be able to use reversion to fall back to a safe mode. This is similar to roll-back recovery but can be a human action if humans are present in the loop.

History

The first known fault-tolerant computer was SAPO, built in 1951 in Czechoslovakia by Antonín Svoboda.[3]:155 Its basic design was magnetic drums connected via relays, with a voting method of memory error detection (triple modular redundancy). Several other machines were developed along this line, mostly for military use. Eventually, they separated into three distinct categories: machines that would last a long time without any maintenance, such as the ones used on NASA space probes and satellites; computers that were very dependable but required constant monitoring, such as those used to monitor and control nuclear power plants or supercollider experiments; and finally, computers with a high amount of runtime which would be under heavy use, such as many of the supercomputers used by insurance companies for their probability monitoring.

Most of the development in the so-called LLNM (Long Life, No Maintenance) computing was done by NASA during the 1960s,[4] in preparation for Project Apollo and other research aspects. NASA's first machine went into a space observatory, and their second attempt, the JSTAR computer, was used in Voyager. This computer had a backup of memory arrays to use memory recovery methods and thus it was called the JPL Self-Testing-And-Repairing computer. It could detect its own errors and fix them or bring up redundant modules as needed. The computer is still working, as of early 2022.[5]

Hyper-dependable computers were pioneered mostly by aircraft manufacturers,[3]:210 nuclear power companies, and the railroad industry in the USA. These needed computers with massive amounts of uptime that would fail gracefully enough with a fault to allow continued operation while relying on the fact that the computer output would be constantly monitored by humans to detect faults. Again, IBM developed the first computer of this kind for NASA for guidance of Saturn V rockets, but later on BNSF, Unisys, and General Electric built their own.[3]:223

In the 1970s, much work has happened in the field .[6][7][8] For instance, F14 CADC had built-in self-test and redundancy.[9]

In general, the early efforts at fault-tolerant designs were focused mainly on internal diagnosis, where a fault would indicate something was failing and a worker could replace it. SAPO, for

instance, had a method by which faulty memory drums would emit a noise before failure.[10] Later efforts showed that to be fully effective, the system had to be self-repairing and diagnosing – isolating a fault and then implementing a redundant backup while alerting a need for repair. This is known as N-model redundancy, where faults cause automatic fail-safes and a warning to the operator, and it is still the most common form of level one fault-tolerant design in use today.

Voting was another initial method, as discussed above, with multiple redundant backups operating constantly and checking each other's results, with the outcome that if, for example, four components reported an answer of 5 and one component reported an answer of 6, the other four would "vote" that the fifth component was faulty and have it taken out of service. This is called M out of N majority voting.

Historically, the motion has always been to move further from N-model and more to M out of N due to the fact that the complexity of systems and the difficulty of ensuring the transitive state from fault-negative to fault-positive did not disrupt operations.

Tandem and Stratus were among the first companies specializing in the design of fault-tolerant computer systems for online transaction processing.

Examples

"M2 Mobile Web", the original mobile web front end of Twitter, later served as fallback legacy version to clients without JavaScript support and/or incompatible browsers until December 2020.

Hardware fault tolerance sometimes requires that broken parts be taken out and replaced with new parts while the system is still operational (in computing known as *hot swapping*). Such a system implemented with a single backup is known as **single point tolerant** and represents the vast majority of fault-tolerant systems. In such systems the mean time between failures should be long enough for the operators to have sufficient time to fix the broken devices (mean time to repair) before the backup also fails. It is helpful if the time between failures is as long as possible, but this is not specifically required in a fault-tolerant system.

Fault tolerance is notably successful in computer applications. Tandem Computers built their entire business on such machines, which used single-point tolerance to create their NonStop systems with uptimes measured in years.

Fail-safe architectures may encompass also the computer software, for example by process replication.

Data formats may also be designed to degrade gracefully. HTML for example, is designed to be forward compatible, allowing Web browsers to ignore new and unsupported HTML entities without causing the document to be unusable. Additionally, some sites, including popular platforms such as Twitter (until December 2020), provide an optional lightweight front end that does not rely on JavaScript and has a minimal layout, to ensure wide accessibility and outreach, such as on game consoles with limited web browsing capabilities.[11][12]

Terminology

An example of graceful degradation by design in an image with transparency. Each of the top two images is the result of viewing the composite image in a viewer that recognises transparency. The bottom two images are the result in a viewer with no support for transparency. Because the transparency mask (center bottom) is discarded, only the overlay (center top) remains; the image on the left has been designed to degrade gracefully, hence is still meaningful without its transparency information.

A highly fault-tolerant system might continue at the same level of performance even though one or more components have failed. For example, a building with a backup electrical generator will provide the same voltage to wall outlets even if the grid power fails.

A system that is designed to fail safe, or fail-secure, or **fail gracefully**, whether it functions at a reduced level or fails completely, does so in a way that protects people, property, or data from injury, damage, intrusion, or disclosure. In computers, a program might fail-safe by executing a graceful exit (as opposed to an uncontrolled crash) in order to prevent data corruption after experiencing an error.[13] A similar distinction is made between "failing well" and "failing badly".

A system that is designed to experience **graceful degradation**, or to **fail soft** (used in computing, similar to "fail safe"[14]) operates at a reduced level of performance after some component failures. For example, a building may operate lighting at reduced levels and elevators at reduced speeds if grid power fails, rather than either trapping people in the dark completely or continuing to operate at full power. In computing an example of graceful degradation is that if insufficient network bandwidth is available to stream an online video, a lower-resolution version might be streamed in place of the high-resolution version. Progressive enhancement is an example in computing, where web pages are available in a basic functional format for older, small-screen, or limited-capability web browsers, but in an enhanced version for browsers capable of handling additional technologies or that have a larger display available.

In fault-tolerant computer systems, programs that are considered robust are designed to continue operation despite an error, exception, or invalid input, instead of crashing completely. Software brittleness is the opposite of robustness. Resilient networks continue to transmit data despite the failure of some links or nodes; resilient buildings and infrastructure are likewise expected to prevent complete failure in situations like earthquakes, floods, or collisions.

A system with high failure transparency will alert users that a component failure has occurred, even if it continues to operate with full performance, so that failure can be repaired or imminent complete failure anticipated.[15] Likewise, a fail-fast component is designed to report at the first point of failure, rather than allow downstream components to fail and generate reports then. This allows easier diagnosis of the underlying problem, and may prevent improper operation in a broken state.

A **single fault condition** is a situation where one means for protection against a hazard is defective. If a single fault condition results unavoidably in another single fault condition, the two

failures are considered as one single fault condition.[16] A source offers the following example:

> A *single-fault condition* is a condition when a single means for protection against hazard in equipment is defective or a single external abnormal condition is present, e.g. short circuit between the live parts and the applied part.[17]

Criteria

Providing fault-tolerant design for every component is normally not an option. Associated redundancy brings a number of penalties: increase in weight, size, power consumption, cost, as well as time to design, verify, and test. Therefore, a number of choices have to be examined to determine which components should be fault tolerant:[18]

- **How critical is the component?** In a car, the radio is not critical, so this component has less need for fault tolerance.
- **How likely is the component to fail?** Some components, like the drive shaft in a car, are not likely to fail, so no fault tolerance is needed.
- **How expensive is it to make the component fault tolerant?** Requiring a redundant car engine, for example, would likely be too expensive both economically and in terms of weight and space, to be considered.

An example of a component that passes all the tests is a car's occupant restraint system. While we do not normally think of the *primary* occupant restraint system, it is gravity. If the vehicle rolls over or undergoes severe g-forces, then this primary method of occupant restraint may fail. Restraining the occupants during such an accident is absolutely critical to safety, so we pass the first test. Accidents causing occupant ejection were quite common before seat belts, so we pass the second test. The cost of a redundant restraint method like seat belts is quite low, both economically and in terms of weight and space, so we pass the third test. Therefore, adding seat belts to all vehicles is an excellent idea. Other "supplemental restraint systems", such as airbags, are more expensive and so pass that test by a smaller margin.

Another excellent and long-term example of this principle being put into practice is the braking system: whilst the actual brake mechanisms are critical, they are not particularly prone to sudden (rather than progressive) failure, and are in any case necessarily duplicated to allow even and balanced application of brake force to all wheels. It would also be prohibitively costly to

further double-up the main components and they would add considerable weight. However, the similarly critical systems for actuating the brakes under driver control are inherently less robust, generally using a cable (can rust, stretch, jam, snap) or hydraulic fluid (can leak, boil and develop bubbles, absorb water and thus lose effectiveness). Thus in most modern cars the footbrake hydraulic brake circuit is diagonally divided to give two smaller points of failure, the loss of either only reducing brake power by 50% and not causing as much dangerous brakeforce imbalance as a straight front-back or left-right split, and should the hydraulic circuit fail completely (a relatively very rare occurrence), there is a failsafe in the form of the cable-actuated parking brake that operates the otherwise relatively weak rear brakes, but can still bring the vehicle to a safe halt in conjunction with transmission/engine braking so long as the demands on it are in line with normal traffic flow. The cumulatively unlikely combination of total foot brake failure with the need for harsh braking in an emergency will likely result in a collision, but still one at lower speed than would otherwise have been the case.

In comparison with the foot pedal activated service brake, the parking brake itself is a less critical item, and unless it is being used as a one-time backup for the footbrake, will not cause immediate danger if it is found to be nonfunctional at the moment of application. Therefore, no redundancy is built into it per se (and it typically uses a cheaper, lighter, but less hardwearing cable actuation system), and it can suffice, if this happens on a hill, to use the footbrake to momentarily hold the vehicle still, before driving off to find a flat piece of road on which to stop. Alternatively, on shallow gradients, the transmission can be shifted into Park, Reverse or First gear, and the transmission lock / engine compression used to hold it stationary, as there is no need for them to include the sophistication to first bring it to a halt.

On motorcycles, a similar level of fail-safety is provided by simpler methods; firstly the front and rear brake systems being entirely separate, regardless of their method of activation (that can be cable, rod or hydraulic), allowing one to fail entirely whilst leaving the other unaffected. Secondly, the rear brake is relatively strong compared to its automotive cousin, even being a powerful disc on sports models, even though the usual intent is for the front system to provide the vast majority of braking force; as the overall vehicle weight is more central, the rear tyre is generally larger and grippier, and the rider can lean back to put more weight on it, therefore allowing more brake force to be applied before the wheel locks up. On cheaper, slower utility-class machines, even if the front wheel should use a hydraulic disc for extra brake force and easier packaging, the rear will usually be a primitive, somewhat inefficient, but exceptionally robust rod-actuated drum, thanks to the ease of connecting the footpedal to the wheel in this way and, more importantly, the near impossibility of catastrophic failure even if the rest of the machine, like a

lot of low-priced bikes after their first few years of use, is on the point of collapse from neglected maintenance.

Requirements

The basic characteristics of fault tolerance require:

1. No single point of failure – If a system experiences a failure, it must continue to operate without interruption during the repair process.
2. Fault isolation to the failing component – When a failure occurs, the system must be able to isolate the failure to the offending component. This requires the addition of dedicated failure detection mechanisms that exist only for the purpose of fault isolation. Recovery from a fault condition requires classifying the fault or failing component. The National Institute of Standards and Technology (NIST) categorizes faults based on locality, cause, duration, and effect.
3. Fault containment to prevent propagation of the failure – Some failure mechanisms can cause a system to fail by propagating the failure to the rest of the system. An example of this kind of failure is the "rogue transmitter" that can swamp legitimate communication in a system and cause overall system failure. Firewalls or other mechanisms that isolate a rogue transmitter or failing component to protect the system are required.
4. Availability of reversion modes

In addition, fault-tolerant systems are characterized in terms of both planned service outages and unplanned service outages. These are usually measured at the application level and not just at a hardware level. The figure of merit is called availability and is expressed as a percentage. For example, a five nines system would statistically provide 99.999% availability.

Fault-tolerant systems are typically based on the concept of redundancy.

Fault tolerance techniques

Research into the kinds of tolerances needed for critical systems involves a large amount of interdisciplinary work. The more complex the system, the more carefully all possible interactions have to be considered and prepared for. Considering the importance of high-value systems in transport, public utilities and the military, the field of topics that touch on research is very wide: it can include such obvious subjects as software modeling and reliability, or hardware design, to

arcane elements such as stochastic models, graph theory, formal or exclusionary logic, parallel processing, remote data transmission, and more.[19]

Replication

Spare components address the first fundamental characteristic of fault tolerance in three ways:

- Replication: Providing multiple identical instances of the same system or subsystem, directing tasks or requests to all of them in parallel, and choosing the correct result on the basis of a quorum;
- Redundancy: Providing multiple identical instances of the same system and switching to one of the remaining instances in case of a failure (failover);
- Diversity: Providing multiple *different* implementations of the same specification, and using them like replicated systems to cope with errors in a specific implementation.

All implementations of RAID, redundant array of independent disks, except RAID 0, are examples of a fault-tolerant storage device that uses data redundancy.

A lockstep fault-tolerant machine uses replicated elements operating in parallel. At any time, all the replications of each element should be in the same state. The same inputs are provided to each replication, and the same outputs are expected. The outputs of the replications are compared using a voting circuit. A machine with two replications of each element is termed dual modular redundant (DMR). The voting circuit can then only detect a mismatch and recovery relies on other methods. A machine with three replications of each element is termed triple modular redundant (TMR). The voting circuit can determine which replication is in error when a two-to-one vote is observed. In this case, the voting circuit can output the correct result, and discard the erroneous version. After this, the internal state of the erroneous replication is assumed to be different from that of the other two, and the voting circuit can switch to a DMR mode. This model can be applied to any larger number of replications.

Lockstep fault-tolerant machines are most easily made fully synchronous, with each gate of each replication making the same state transition on the same edge of the clock, and the clocks to the replications being exactly in phase. However, it is possible to build lockstep systems without this requirement.

Bringing the replications into synchrony requires making their internal stored states the same. They can be started from a fixed initial state, such as the reset state. Alternatively, the internal

state of one replica can be copied to another replica.

One variant of DMR is **pair-and-spare**. Two replicated elements operate in lockstep as a pair, with a voting circuit that detects any mismatch between their operations and outputs a signal indicating that there is an error. Another pair operates exactly the same way. A final circuit selects the output of the pair that does not proclaim that it is in error. Pair-and-spare requires four replicas rather than the three of TMR, but has been used commercially.

Failure-oblivious computing

Failure-oblivious computing is a technique that enables computer programs to continue executing despite errors.[20] The technique can be applied in different contexts. First, it can handle invalid memory reads by returning a manufactured value to the program,[21] which in turn, makes use of the manufactured value and ignores the former memory value it tried to access, this is a great contrast to typical memory checkers, which inform the program of the error or abort the program. Second, it can be applied to exceptions where some catch blocks are written or synthesized to catch unexpected exceptions.[22] Furthermore, it happens that the execution is modified several times in a row, in order to prevent cascading failures.[23]

The approach has performance costs: because the technique rewrites code to insert dynamic checks for address validity, execution time will increase by 80% to 500%.[24]

Recovery shepherding

Recovery shepherding is a lightweight technique to enable software programs to recover from otherwise fatal errors such as null pointer dereference and divide by zero.[25] Comparing to the failure oblivious computing technique, recovery shepherding works on the compiled program binary directly and does not need to recompile to program.

It uses the just-in-time binary instrumentation framework Pin. It attaches to the application process when an error occurs, repairs the execution, tracks the repair effects as the execution continues, contains the repair effects within the application process, and detaches from the process after all repair effects are flushed from the process state. It does not interfere with the normal execution of the program and therefore incurs negligible overhead.[25] For 17 of 18 systematically collected real world null-dereference and divide-by-zero errors, a prototype implementation enables the application to continue to execute to provide acceptable output and service to its users on the error-triggering inputs.[25]

Circuit breaker

The circuit breaker design pattern is a technique to avoid catastrophic failures in distributed systems.

Redundancy

Redundancy is the provision of functional capabilities that would be unnecessary in a fault-free environment.[26] This can consist of backup components that automatically "kick in" if one component fails. For example, large cargo trucks can lose a tire without any major consequences. They have many tires, and no one tire is critical (with the exception of the front tires, which are used to steer, but generally carry less load, each and in total, than the other four to 16, so are less likely to fail). The idea of incorporating redundancy in order to improve the reliability of a system was pioneered by John von Neumann in the 1950s.[27]

Two kinds of redundancy are possible:[28] space redundancy and time redundancy. Space redundancy provides additional components, functions, or data items that are unnecessary for fault-free operation. Space redundancy is further classified into hardware, software and information redundancy, depending on the type of redundant resources added to the system. In time redundancy the computation or data transmission is repeated and the result is compared to a stored copy of the previous result. The current terminology for this kind of testing is referred to as 'In Service Fault Tolerance Testing or ISFTT for short.

Disadvantages

Fault-tolerant design's advantages are obvious, while many of its disadvantages are not:

- **Interference with fault detection in the same component.** To continue the above passenger vehicle example, with either of the fault-tolerant systems it may not be obvious to the driver when a tire has been punctured. This is usually handled with a separate "automated fault-detection system". In the case of the tire, an air pressure monitor detects the loss of pressure and notifies the driver. The alternative is a "manual fault-detection system", such as manually inspecting all tires at each stop.
- **Interference with fault detection in another component.** Another variation of this problem is when fault tolerance in one component prevents fault detection in a different component. For example, if component B performs some operation based on the output from component A,

then fault tolerance in B can hide a problem with A. If component B is later changed (to a less fault-tolerant design) the system may fail suddenly, making it appear that the new component B is the problem. Only after the system has been carefully scrutinized will it become clear that the root problem is actually with component A.

- **Reduction of priority of fault correction.** Even if the operator is aware of the fault, having a fault-tolerant system is likely to reduce the importance of repairing the fault. If the faults are not corrected, this will eventually lead to system failure, when the fault-tolerant component fails completely or when all redundant components have also failed.
- **Test difficulty.** For certain critical fault-tolerant systems, such as a nuclear reactor, there is no easy way to verify that the backup components are functional. The most infamous example of this is Chernobyl, where operators tested the emergency backup cooling by disabling primary and secondary cooling. The backup failed, resulting in a core meltdown and massive release of radiation.
- **Cost.** Both fault-tolerant components and redundant components tend to increase cost. This can be a purely economic cost or can include other measures, such as weight. Manned spaceships, for example, have so many redundant and fault-tolerant components that their weight is increased dramatically over unmanned systems, which don't require the same level of safety.
- **Inferior components.** A fault-tolerant design may allow for the use of inferior components, which would have otherwise made the system inoperable. While this practice has the potential to mitigate the cost increase, use of multiple inferior components may lower the reliability of the system to a level equal to, or even worse than, a comparable non-fault-tolerant system.

Related terms

There is a difference between fault tolerance and systems that rarely have problems. For instance, the Western Electric crossbar systems had failure rates of two hours per forty years, and therefore were highly *fault resistant*. But when a fault did occur they still stopped operating completely, and therefore were not *fault tolerant*.

See also

- Byzantine fault tolerance
- Control reconfiguration

- Damage tolerance
- Data redundancy
- Defence in depth
- Ecological resilience
- Elegant degradation
- Error detection and correction
- Error-tolerant design (human error-tolerant design)
- Fail-safe
- Failure semantics
- Fall back and forward
- Graceful exit
- Intrusion tolerance
- List of system quality attributes
- Progressive enhancement
- Resilience (network)
- Robustness (computer science)
- Rollback (data management)
- Self-management (computer science)
- Crash-only software

References

1. *Adaptive Fault Tolerance and Graceful Degradation (http://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1186&context=cs_faculty_pubs)* , Oscar González et al., 1997, University of Massachusetts - Amherst
2. *Johnson, B. W. (1984). "Fault-Tolerant Microprocessor-Based Systems (https://www.computer.org/csdl/mags/mi/1984/06/04071150.pdf)* ", IEEE Micro, vol. 4, no. 6, pp. 6–21
3. *Daniel P. Siewiorek; C. Gordon Bell; Allen Newell (1982). Computer Structures: Principles and Examples (https://archive.org/details/computerstructur01siew)* . *McGraw-Hill. ISBN 0-07-057302-6.*

4. Algirdas Avižienis; George C. Gilley; Francis P. Mathur; David A. Rennels; John A. Rohr; David K. Rubin. "The STAR (Self-Testing And Repairing) Computer: An Investigation Of the Theory and Practice Of Fault-tolerant Computer Design" (https://www.computer.org/csdl/proceedings/ftcsh/1995/7150/00/00532604.pdf) (PDF).

5. "Voyager Mission state (more often than not at least three months out of date)" (https://voyager.jpl.nasa.gov/mission/weekly-reports/) . NASA. Retrieved 2022-04-01.

6. Randell, Brian; Lee, P.A.; Treleaven, P. C. (June 1978). "Reliability Issues in Computing System Design" (http://portal.acm.org/citation.cfm?id=356729&coll=&dl=ACM&CFID=15151515&CFTOKEN=6184618) . ACM Computing Surveys. **10** (2): 123–165. doi:10.1145/356725.356729 (https://doi.org/10.1145%2F356725.356729) . ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300) . S2CID 16909447 (https://api.semanticscholar.org/CorpusID:16909447) .

7. P. J. Denning (December 1976). "Fault tolerant operating systems" (http://portal.acm.org/citation.cfm?id=356680&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618) . ACM Computing Surveys. **8** (4): 359–389. doi:10.1145/356678.356680 (https://doi.org/10.1145%2F356678.356680) . ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300) . S2CID 207736773 (https://api.semanticscholar.org/CorpusID:207736773) .

8. Theodore A. Linden (December 1976). "Operating System Structures to Support Security and Reliable Software" (http://portal.acm.org/citation.cfm?id=356682&coll=&dl=ACM&CFID=15151515&CFTOKEN=61784618) . ACM Computing Surveys. **8** (4): 409–445. doi:10.1145/356678.356682 (https://doi.org/10.1145%2F356678.356682) . hdl:2027/mdp.39015086560037 (https://hdl.handle.net/2027%2Fmdp.39015086560037) . ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300) . S2CID 16720589 (https://api.semanticscholar.org/CorpusID:16720589) .

9. Ray Holt. "The F14A Central Air Data Computer, and the LSI Technology State-of-the-Art in 1968" (http://www.firstmicroprocessor.com/documents/lsistate-97.pdf) .

10. Fault tolerant computing in computer design Neilforoshan, M.R Journal of Computing Sciences in Colleges archive Volume 18, Issue 4 (April 2003) Pages: 213 – 220, ISSN 1937-4771 (https://www.worldcat.org/search?fq=x0:jrnl&q=n2:1937-4771)

11. "Why your website should work without JavaScript" (https://dev.to/shadowfaxrodeo/why-your-website-should-work-without-javascript-3kko) . DEV Community. Retrieved 2021-05-16.

12. Fairfax, Zackerie (2020-11-28). "Legacy Twitter Shutdown Means You Can't Tweet From The 3DS Anymore" (https://screenrant.com/twitter-legacy-nintendo-3ds-shut-down-date-december-2020/) . ScreenRant. Retrieved 2021-07-01.

13. Hudak, J.J.; Suh, B.-H.; Siewiorek, D.P.; Segall, Z. (1993). "Evaluation and comparison of fault-tolerant software techniques" (https://ieeexplore.ieee.org/abstract/document/229487) . IEEE Transactions on Reliability. **42** (2): 190–204. doi:10.1109/24.229487 (https://doi.org/10.1109%2F24.229487) . ISSN 1558-1721 (https://www.worldcat.org/issn/1558-1721) .

14. *Stallings, W (2009): Operating Systems. Internals and Design Principles, sixth edition*

15. *Thampi, Sabu M. (2009-11-23). "Introduction to Distributed Systems". arXiv:0911.4395 (https://arxiv.org/abs/0911.4395) [cs.DC (https://arxiv.org/archive/cs.DC)].*

16. *"Control" (https://web.archive.org/web/19991008210224/http://grouper.ieee.org/groups/1461/glossary.htm) . Grouper.ieee.org. Archived from the original (http://grouper.ieee.org/groups/1461/glossary.htm) on 1999-10-08. Retrieved 2016-04-06.*

17. *Baha Al-Shaikh, Simon G. Stacey, Essentials of Equipment in Anaesthesia, Critical Care, and Peri-Operative Medicine (2017), p. 247.*

18. *Dubrova, E. (2013). "Fault-Tolerant Design", Springer, 2013, ISBN 978-1-4614-2112-2*

19. *Reliability evaluation of some fault-tolerant computer architectures. Springer-Verlag. November 1980. ISBN 978-3-540-10274-8.*

20. *Herzberg, Amir; Shulman, Haya (2012). "Oblivious and Fair Server-Aided Two-Party Computation" (https://dx.doi.org/10.1109/ares.2012.28) . 2012 Seventh International Conference on Availability, Reliability and Security. IEEE: 75–84. doi:10.1109/ares.2012.28 (https://doi.org/10.1109%2Fares.2012.28) . ISBN 978-1-4673-2244-7. S2CID 6579295 (https://api.semanticscholar.org/CorpusID:6579295) .*

21. *Rigger, Manuel; Pekarek, Daniel; Mössenböck, Hanspeter (2018), "Context-Aware Failure-Oblivious Computing as a Means of Preventing Buffer Overflows" (https://dx.doi.org/10.1007/978-3-030-02744-5_28) , Network and System Security, Cham: Springer International Publishing, pp. 376–390, arXiv:1806.09026 (https://arxiv.org/abs/1806.09026) , doi:10.1007/978-3-030-02744-5_28 (https://doi.org/10.1007%2F978-3-030-02744-5_28) , ISBN 978-3-030-02743-8, retrieved 2020-10-07*

22. *Zhang, Long; Monperrus, Martin (2019). "TripleAgent: Monitoring, Perturbation and Failure-Obliviousness for Automated Resilience Improvement in Java Applications" (https://arxiv.org/pdf/1812.10706) . 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). Berlin, Germany: IEEE: 116–127. arXiv:1812.10706 (https://arxiv.org/abs/1812.10706) . doi:10.1109/ISSRE.2019.00021 (https://doi.org/10.1109%2FISSRE.2019.00021) . ISBN 978-1-7281-4982-0. S2CID 57189195 (https://api.semanticscholar.org/CorpusID:57189195) .*

23. *Durieux, Thomas; Hamadi, Youssef; Yu, Zhongxing; Baudry, Benoit; Monperrus, Martin (2018). "Exhaustive Exploration of the Failure-Oblivious Computing Search Space". 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). pp. 139–149. arXiv:1710.09722 (https://arxiv.org/abs/1710.09722) . doi:10.1109/ICST.2018.00023 (https://doi.org/10.1109%2FICST.2018.00023) . ISBN 978-1-5386-5012-7. S2CID 4304123 (https://api.semanticscholar.org/CorpusID:4304123) .*

24. *Keromytis, Angelos D. (2007), "Characterizing Software Self-Healing Systems" (https://books.google.com/books?id=N2uIjckxHSoC&q=failure-oblivious&pg=PA27) , in Gorodetski, Vladimir I.; Kotenko, Igor; Skormin, Victor A. (eds.), Characterizing Software Self-Healing Systems, Computer network security: Fourth International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, Springer, ISBN 978-3-540-73985-2*

25. *Long, Fan; Sidiroglou-Douskos, Stelios; Rinard, Martin (2014). "Automatic Runtime Error Repair and Containment via Recovery Shepherding". Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI '14'. New York, NY, USA: ACM. pp. 227–238. doi:10.1145/2594291.2594337 (https://doi.org/10.1145%2F2594291.2594337) . ISBN 978-1-4503-2784-8. S2CID 6252501 (https://api.semanticscholar.org/CorpusID:6252501) .*

26. *Laprie, J. C. (1985). "Dependable Computing and Fault Tolerance: Concepts and Terminology (http://www.macedo.ufba.br/conceptsANDTermonology.pdf) ", Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15), pp. 2–11*

27. *von Neumann, J. (1956). "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components (http://www.cyclify.com/wiki/images/a/af/Von_Neumann_Probabilistic_Logics_and_the_Synthesis_of_Reliable_Organisms_from_Unreliable_Components.pdf) ", in Automata Studies, eds. C. Shannon and J. McCarthy, Princeton University Press, pp. 43–98*

28. *Avizienis, A. (1976). "Fault-Tolerant Systems (https://www.computer.org/csdl/trans/tc/1976/12/01674598.pdf) ", IEEE Transactions on Computers, vol. 25, no. 12, pp. 1304–1312*

Last edited 22 days ago by Martin.monperrus