Distributed Algorithms

Fault Tolerance

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Architecture of Application Systems
Faculty for Computer Science and Electrical Engineering
University of Rostock

Overview

> Introduction into fault tolerance (this lecture)

> Masking fault tolerance (next lecture)
>> Byzantines Generals

Fault Tolerance

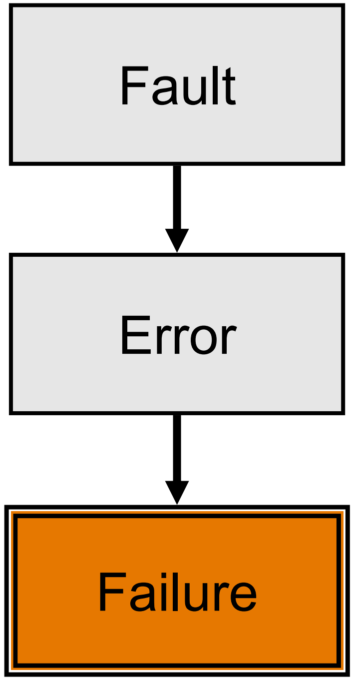
- Every (non-trivial) system contains faults!
- Taking faults into account is, thus, absolutely necessary!
- In large systems (e.g., the Internet) some components will always be faulty!

- Simple motivation: all computers of a systems must be available at the same time (→ serial composition)
  - 1 computer → system unavailable 1% of the time
  - 10 computers → system unavailable 10% of the time
  - 100 computers → system unavailable 63% of the time
  - 1000 computers → system unavailable 99.99% of the time

- Example: How often are all computers in a large pool room functional at the same time?

Basic Terms

- Fault
  - Triggering event, e.g., caused by external disturbance or abrasion
- Error
  - Internal system state violating the specification, caused by a fault
- Failure
  - System does not provide the correct service to the outside, caused by an error
- This chain might abort after every step, a failure is, thus, not always the consequence



Classification of Faults – Benign Faults

- Crash fault
  - A node suddenly fails and afterwards no actions are executed; until the moment the node fails, it behaves according to its specification
  - Also denoted halting failure or fail stop of the node
  - Often, those terms additionally mean that the node failure can be surely determined by the correct nodes
- Omission fault
  - Node does not execute some actions that should be executed
- Timing fault
  - Node behaves correctly, but executes some actions too late
- All faults above have in common that the respective process does not execute actions a correct process would not execute

⇒ Assuming these benign faults only is not always realistic!

Classification of Faults – Malicious Faults

- Often denoted as Byzantine faults (Lamport, 1982)
- Means that faulty processes can execute arbitrary actions and can also cooperate among each other
  - E.g., execute arbitrary calculations and send arbitrary messages
- Often, the model is restricted such that processes can only execute actions computable with polynomial time complexity
  - This means, e.g., that faulty processes cannot fake digital signatures of correct nodes
- Model covers all kinds of faults, e.g., it also includes attacks on a system from outside
- Model also contains all “simpler” cases

Fault Avoidance vs. Fault Tolerance

- Fault avoidance
  - Idea: “Avoid faults!”
  - Eliminating fault causes
    - Very reliable components
    - Extensive testing
    - …
- Fault tolerance
  - Idea: “Faults occur, tolerate them!”

  → Considered here

Paradigms of Fault Tolerance

- Masking fault tolerance
  - Aim is to avoid system failure (if possible)
- Non-masking fault tolerance
  - System may fail partly or temporarily
  - Better than a complete and/or permanent failure

Masking Fault Tolerance

- Necessary if a (even only temporary) failure of the system would have inacceptable consequences
  - e.g., death of humans or high financial losses
- Tries to ensure safety and liveness

- Example: car brakes
  - Separate brake circuits for right front wheel and left back wheel as well as for left front wheel and right back wheel
  - Car can still break if one circuit has failed

Masking Fault Tolerance

> Always needs redundancy for implementation
> Always only possible for the faults considered
> Can never take into account all possible faults
> Can only be successful if solely a limited part of the system components fails
> The amount of the failed components that can be tolerated depends on the fault model

Redundancy in Space or Time

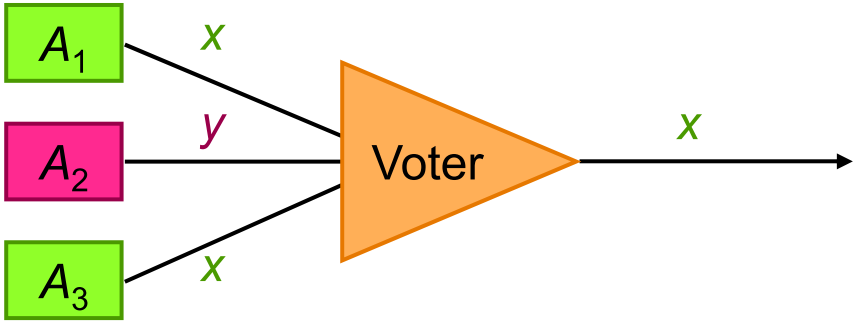
- Redundancy in Space:
  multiple instances of components
  - example: a server has several independent power supplies
    - single, functioning power supply sufficient
    - failed power supplies are substituted without disrupting server operation
- Redundancy in Time:
  multiple execution of actions
  - example: packages are sent several times over an unreliable network; from the received packages, an error-free package is generated (if possible)

Active Replication vs. Passive Replication

- Active Replication
  - All replicas collaborate productively
    - e.g. eyes, aircraft engines, brake systems,...
- Passive Replication
  - Replicas only get active in case of a failure (e.g., *primary/backup*)
  - Example: emergency power generator

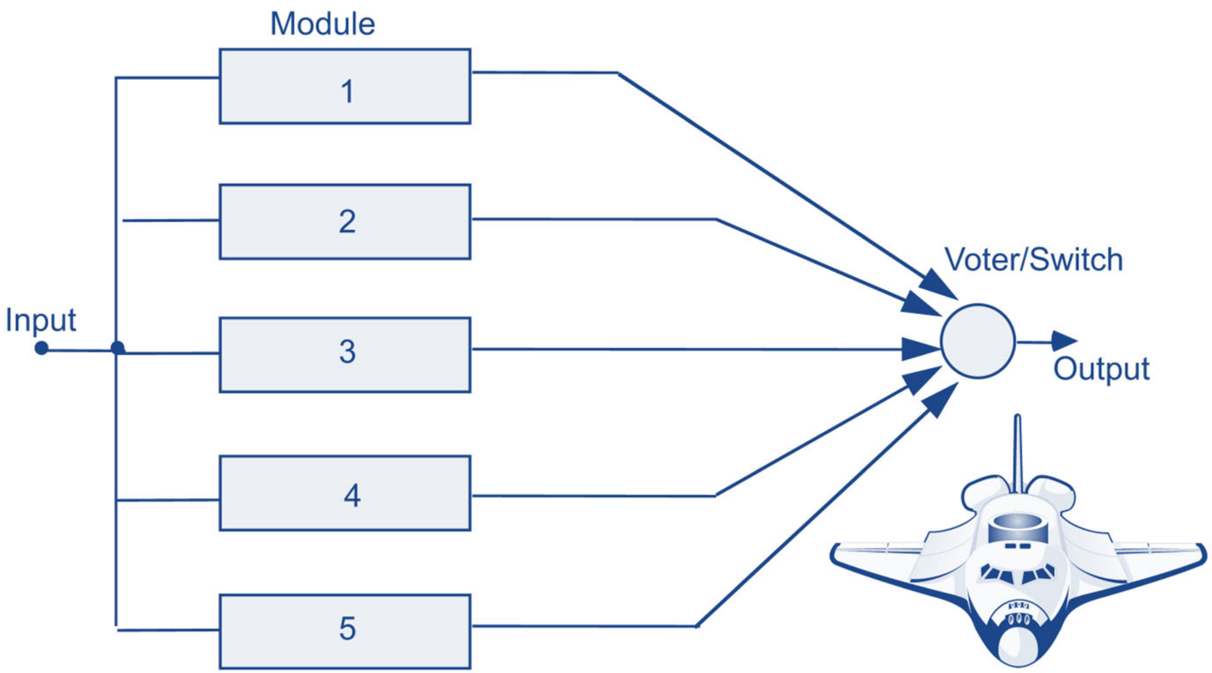
Triple Modular Redundancy (TMR)

- Example for a mechanism using active replication
- Three identical replicas execute the same computation and generate a result
- Highly reliable voter compares the three results and performs a majority vote to deliver the correct results
- As long as only one of the replicas fails, the output is correct as long as the voter has not failed
- Example: three pre-cogs in Minority Report lead to a conviction even if one is in the minority



Example: Space Shuttle

- TMR with two stand-by modules
  - Modules 1, 2, and 3: active
  - Module 4: warm standby
  - Module 5: cold standby



Fault Tolerance by Redundant Components

- A system is *k-reliable* with respect to a component existing *n* times in the system, if the system can tolerate the failure of up to *k* of the *n* instances of the components
- Non-byzantine faults
  - System with component this is replicated *n*-times is (n – 1)-reliable with respect to that component
- Byzantine faults
  - Assumption: correct output can uniquely be determined by highly reliable voter from the *n* outputs
  - System with a component that is replicated *n*-times is $\left\lfloor \frac{n-1}{2} \right\rfloor$ reliable with respect to that component

Example: 1-Reliability

- Consider a file server *S* with the operations *read*() and *write*()
- Non-byzantine faults
  - *S* sends in response to *read*() either correct content or nothing
  - Realization with two servers *S*1 and *S*2
    - *write*(): to both servers
    - *read*(): if *S*1 does not reply, then *S*2
      also possible: ask *S*1 and *S*2; use first answer
- Byzantine faults
  - *S* sends in response to *read*() either the correct content or a false content or nothing
  - Realization with three servers *S*1, *S*2 and *S*3
    - *write*(): to all three servers
    - *read*(): to all three servers, majority vote at the client

N-Version Programming

- Application is implemented several times by *independent* programmer teams
- Results are evaluated by majority vote at run time
- Problems
  - Results with admissible inaccuracy (e.g., result of a numeric approximation)
  - Multiple possible correct solutions (e.g., zero of a polynomial)
  - Development often not really “independent”
  - …

Problem: Hidden Dependencies

- Hidden dependencies of redundant instances increase probability of simultaneous failure!
- Examples
  - Redundant power supplies in the same electric circuit
  - Redundant servers cooled by the same air condition
  - Redundant diesel generators sharing the same diesel tank
  - Programming teams with same education in N-version programming
  - Same, faulty compiler in N-version programming
  - Redundant computation centers in the same area
  - …

Non-Masking Fault Tolerance

- Applicable if a partial or temporary system failure is acceptable
- 1st possible goal: bring the system into a safe state (fail safe)
  - Assures safety
  - Example: mechanic stop sign for trains
    - When the signal rope is torn, the arm of the signal falls into the position “Stop!”

Quelle: Deutsche Fotothek

Non-Masking Fault Tolerance

- 2nd possible goal: keep the system running, but with restricted functionality (graceful degradation)
  - Assures safety and limited liveness
  - Example: servo steering
    - In case of the failure of the servo pump, steering is still possible, but only with higher effort
  - Example: ABS brake
    - In case of a failure of the ABS device, the brake is still operational without ABS functionality

Non-Masking Fault Tolerance

- 3rd possible goal: reconfigure the system such that it works correctly again
  - Assures safety after reconfiguration again (eventual safety)
  - Liveness is not affected if the reconfiguration only lasts for a limited time
  - Example: communication over serially connected lines
    - In case of the failure of one or several lines, an alternative route is set up without using the failed lines

Exemplary Exam Questions

1. Define the terms Fault, Error and Failure!
2. Explain the difference between benign and malicious faults!
3. What is the difference between masking and non-masking fault tolerance?
4. What is meant by „redundancy in space“ and „redundancy in time“?
5. Explain the terms Fail Safe, Graceful Degradation and N-Version-Programming!

Literature

1. T. Anderson, P.A. Lee: Fault Tolerance – Principles and Practice, Prentice Hall, 1982
2. D.K. Pradhan (Hrsg.): Fault Tolerant Computer Systems, Prentice Hall, 1996
3. D.P. Siewiorek, R.S. Swarz: The Theory and Practice of Reliable Systems Design, Digital Press, 1995

…and many more

Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

`gero.muehl@uni-rostock.de`
`http://wwwava.informatik.uni-rostock.de`