Distributed Algorithms 2015/16

Fault Tolerance

Reinhardt Karnapke | Communication and Operating Systems Group

Overview

- Introduction into fault tolerance (this lecture)
- Masking fault tolerance (next lecture)
  - Redundancy techniques (only sketched, c.f. lecture “Dependable Systems”)
  - Consensus and related problems
- Non-masking fault tolerance (the next but one lecture)
  - Self-Stabilization

Fault Tolerance

- No (non-trivial) system contains no fault!
- Taking faults into account is, thus, absolutely necessary!
- In large systems (e.g., the Internet) some components will always work 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 university computer room: How often are all computers in a large room functional at the same time?

Basic Terms

Fault

- Triggering event, e.g., caused by external disturbance or wear/abrasion

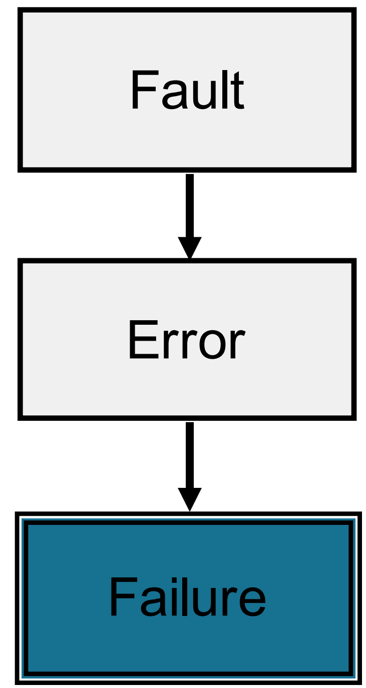
Error

- Internal system state that does not fit 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 not always the consequence



Classification of Faults– Benign Faults

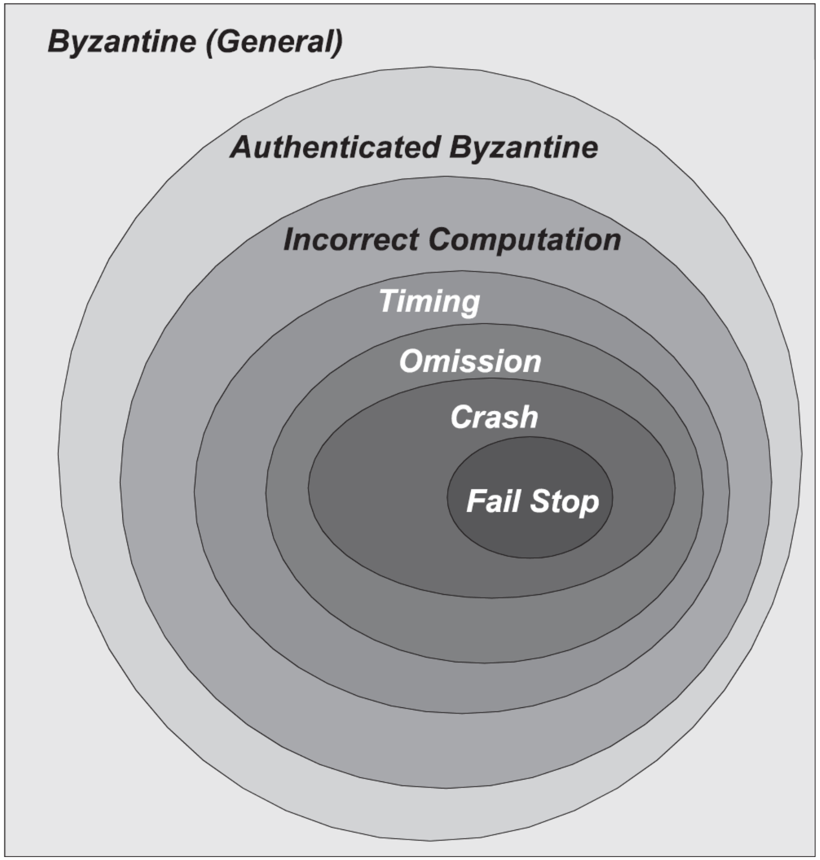
- **Crash Fault**: A node suddenly fails and afterwards no actions are exercised, but until the moment of the fault it behaves according to its specification,
- Also denoted **Halting fault** or **Fail Stop** etc. Often, those terms additionally mean that the fault can be surely determined by all nodes that have not failed
- **Omission Fault**: Some actions are not exercised
- **Timing Fault**: Specification suitable behavior, but some actions are carried out too late
- All have in common: The faulty process does not exercise any actions that a correct process would not do

⇒ Not always realistic!

Classification of Faults– Malicious Faults

- Often denoted as **Byzantine faults** (Lamport, 1982)
- Faulty processes can exercise arbitrary actions and cooperate among each other
  - For example, exercise arbitrary calculations and send arbitrary messages
- Often, also limited model
  - E.g., faulty processes can exercise arbitrary actions calculable with polynomial complexity (i.e., they cannot fake digital signatures of correct nodes)
- Model covers all kinds of faults, e.g., it also targets attacks on a system from outside
- Model also contains all “simpler” cases

Example: Fault models for Distributed Systems



Approaches

- Fault intolerance
  - Idea: “Avoid faults!”
  - Eliminating fault causes
    - Very reliable components
    - Extensive testing
    - …
- Fault tolerance
  - Idea: “Faults occur, we tolerate them!”
  - Fault tolerance by redundancy in space and/or time
  - Acceptance of partial or temporal failures
  - → Considered here

Paradigms of Fault Tolerance

- **Masking fault tolerance**
  - Aim: 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 (also temporary) failure of the system would have inacceptable consequences
  - e.g., death of humans or high financial losses
- Sensible also with many other applications
- Tries to ensure safety and liveness!

- Example car brakes
  - Separated 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
- Only successful if only a limited part of the components is erroneous
- The amount of the erroneous components that can be tolerated depends on the **fault model**

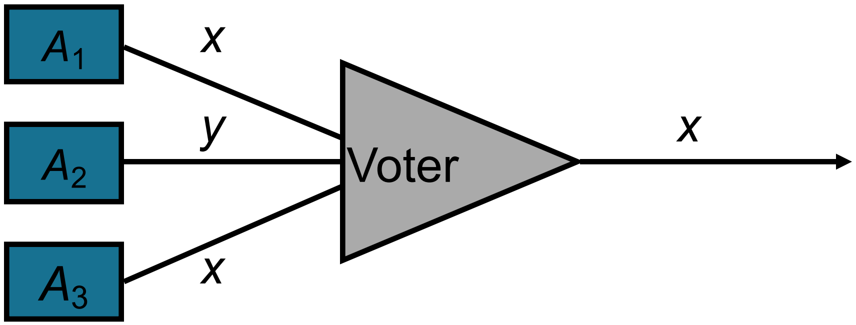
Redundancy in Space or Time

- **Redundancy in Space**:
  multiple instances of components
  - example: a server has several independent power supplies
    - single, functioning power supply sufficient
    - erroneous power supplies are substituted without disrupting server operation

- **Redundancy in Time**:
  multiple execution of actions
  - example: packages are submitted several times over an unreliable network; from those packages, an error free package is generated (if possible)

Active and Passive Replication

- **Active Replication**
  - All replicas collaborate productively
    - e.g. eyes, aircraft engines, brake systems,...
  - TMR (*Triple Modular Redundancy*)



- **Passive Replication**
  - Replicas only get active in case of an error (*primary/backup*)
  - e.g., spare tire (no masking), emergency power generator

Fault Tolerance by Redundant Components

- ***k*-reliability** of a system with respect to a component that exists n times: system can tolerate the failure of up to k of the n instances of the components

- Non-Byzantine fault
  - System with an n-times existing component is $(n-1)$-reliable in reference to that component

- Byzantine fault
  - Assumption: correct output can uniquely be determined from the n outputs by highly reliable voter
  - System with an n-times existing component is

    $\left\lfloor \frac{n-1}{2} \right\rfloor$ reliable in reference to that component

Example: 1–Reliability

- Consider a file server *S* with the operations *read*() and *write*()
- Non-byzantine fault
  - *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 fault
  - *S* sends either the correct content or a false content or nothing after read() is called
  - 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

Example: *N*-Version Programming

- Application is implemented several times by *independent* programmer teams
- Results are evaluated by majority vote at run time
- Problems
  - Comparison of results with admissible inaccuracy (e.g., result of a numeric approximation)
  - Comparison of 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 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 failure of the system/the component is acceptable
- 1. possible aim: In case of an error, bring the system into a safe state (**fail safe**)
  - Assures safety
  - Example: traffic light control
    - In case of an error of the traffic light control, all traffic lights are switched to red
  - Example: mechanic stop sign for trains
    - When the signal rope is torn, the arm of the signal falls into the position “Stop!”

Non-Masking Fault Tolerance

2. possible aim: in case of a failure, the system is run on 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 electronic failure, the brake is still operational without ABS functionality

Non-Masking Fault Tolerance

3. possible aim: In case of a failure, reconfigure the system in such a way that it works correctly again
   - Assures safety again after reconfiguration (**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 usage of the failed lines

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