Distributed Systems

Summer Term 2020

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: July 6, 2020

Distributed Systems

Summer Term 2020

11 Fault Tolerance

Contents

Literature

- Tanenbaum, van Steen: Ch. 7

11.1 Introduction

Concepts

- **Failure**: external incorrect behavior (system no longer keeps its promises)
- **Error**: (unobserved) incorrect internal state
- **Fault**: physical defect (in HW or SW) causing the error
  - fault can be transient, periodic or permanent
- **Fault tolerance**: system does not fail despite a fault
- Requirement for reliable systems:
  - **availability**: p(system is working at time t)
  - **reliability**: p(system is working in time interval Δt)
  - **safety**: no major damage if system fails
  - **maintainability**: effort for “repair” after a failure

Notes for slide 309:

Note the subtle difference between availability and reliability:

- A system that fails every 10 minutes for one millisecond is highly available (99.998%), but very unreliable.
- A system that never fails but must be serviced once a year for 2 weeks is highly reliable, but has an availability of only 96%.

11.1 Introduction ...

Failure models

Crash failure	Server halts
Omission failure	Server is not responding to requests
Receive omission	Server doesn't receive incoming requests
Send omission	Server doesn't send messages
Timing failure	Response time is outside the specification
Response failure	Server's response is incorrect
Value failure	Only the value of the answer is wrong
State transition f.	Incorrect control flow in server
Byzantine failure	Random answers at arbitrary time

- Further distinction: can the client detect the failure or not?

Notes for slide 310:

The term *failure* ist used here, because we are talking about a misbehavior of a server process that is actually visible to a client.

Since a distributed system as a whole coniststs of several clients and servers, fault tolerance also includes the tolerance against failures in a part of the system (e.g., the server). Or, in other words: a failure of a system component (in the sense, that another component will notice the misbehavior) should not lead to a failure of the complete (distributed) system.

11.1 Introduction ...

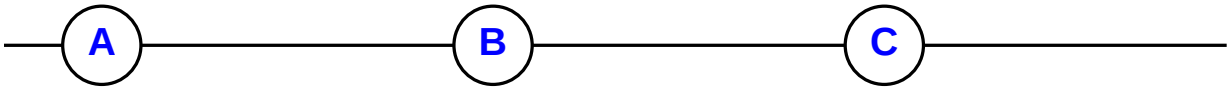
Failure masking through redundancy

- Fault tolerant system must hide faults from other processes
- Most important technique: **redundancy**
  - **information redundancy**: additional “check bits” (e.g., CRC)
  - **time redundancy**: repetition of faulty actions
  - **physical redundancy**: important components are provided multiple times
- Example: **TMR, *triple modular redundancy***
  - components are replicated three times
  - majority decision for the results
  - protects against failure of a replicated component

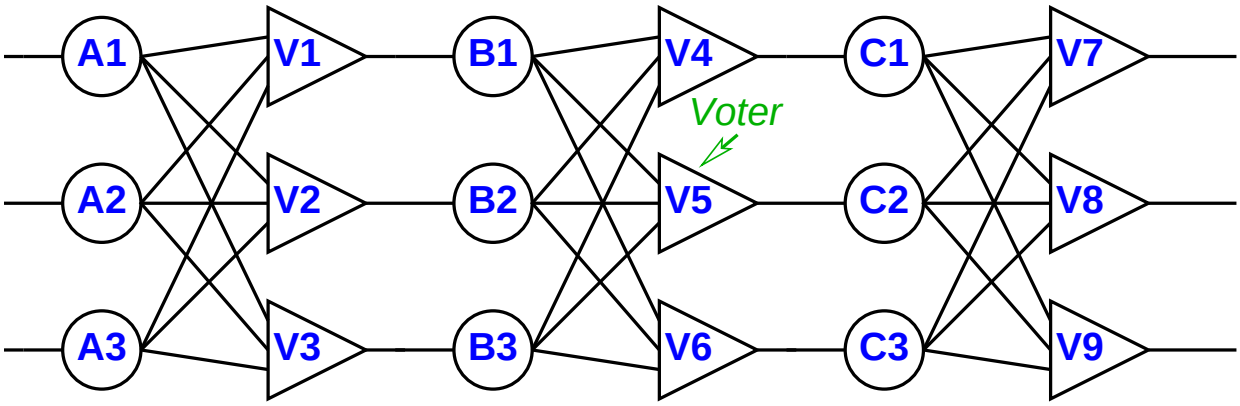
(Animated slide)

Example for TMR

Without redundancy



With TMR



11.2 Process Elasticity

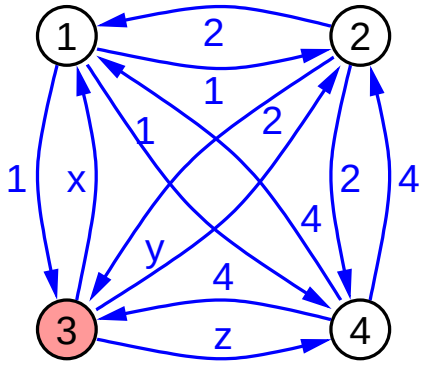
Objective: Protection Against Process Failure

- By replicating processes in groups
  - message to the group is received by all members
    - usually with totally ordered multicast
- Questions:
  - organization of the groups?
    - flat (symmetrical) vs. hierarchical (central coordinator)
    - group administration, synchronous join / exit
  - necessary number of replicas?
    - **k fault tolerant**: failure of k processes can be tolerated
    - for silent failures: $\geq k+1$ Processes
    - for Byzantine failures: $\geq 2k+1$ processes
  - agreement in faulty systems?

Agreement in faulty systems

- Agreement is impossible with unreliable communication
  - two army problem
- Agreement of faulty processes with reliable communication
  - Byzantine agreement problem (*byzantinische Generäle*)
  - agreement only possible if $> \frac{2}{3}$ of the processes work correctly

1. Send information



2. Received information

1: (1, 2, x, 4)
2: (1, 2, y, 4)
3: (1, 2, 3, 4)
4: (1, 2, z, 4)

3. Send received information to all other processes

	1 gets:	2 gets:	4 gets:
from 1:		(1, 2, x ,4)	(1, 2, x ,4)
from 2:	(1, 2, y ,4)		(1, 2, y ,4)
from 3:	(a, b, c, d)	(e, f, g, h)	(i, j, k, l)
from 4:	(1, 2, z ,4)	(1, 2, z ,4)	

Notes for slide 314:

In the two-army problem, two parts of an army must agree on the time for an attack, since they can only win together over the other army:

Generals A and B can only communicate via messengers that can be intercepted, i.e. may not arrive.

- If A suggests an attack time, he doesn't know whether B has received this message. So he doesn't know if B is attacking and therefore won't attack.
- Even if B returns an acknowledgement, he doesn't know if A has received it. So he doesn't know if A is attacking and therefore won't attack.
- Even if A confirms the confirmation again, he does not know whether B has received this confirmation. So he doesn't know if B attacks and therefore won't attack.
- ...

11.3 Reliable Communication

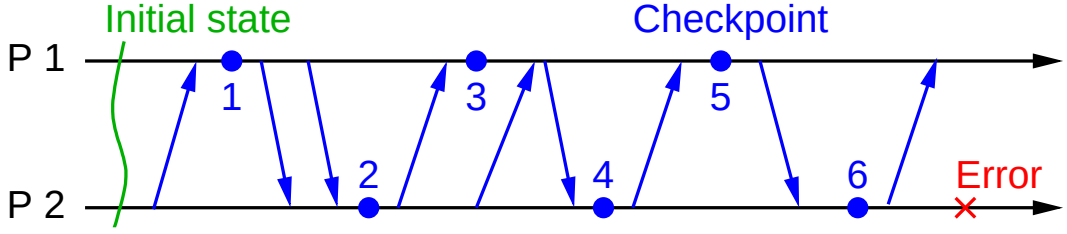
Objective: Protection Against Communication Failures

- Point-to-point communication (☞ **RN_I**)
  - TCP masks omission failures, but not crash failures
- Client/server communication (☞ **2.1**)
  - possible failures:
    - server not found
    - lost request
    - server crash while processing the request
    - lost reply
    - client crash after sending the request
- Group communication (☞ **7.3**)
- Distributed commit (☞ **7.4**)

11.4 Recovery

Objective: System Recovery After an Error

- Forward error recovery: go to a correct new state
- Backward error recovery: go to a correct earlier state
  - i.e. reset to a consistent cut
  - regular backup to stable storage (*checkpointing*)
- Independent checkpointing
  - processes save their state independently of each other
  - problem: domino effect

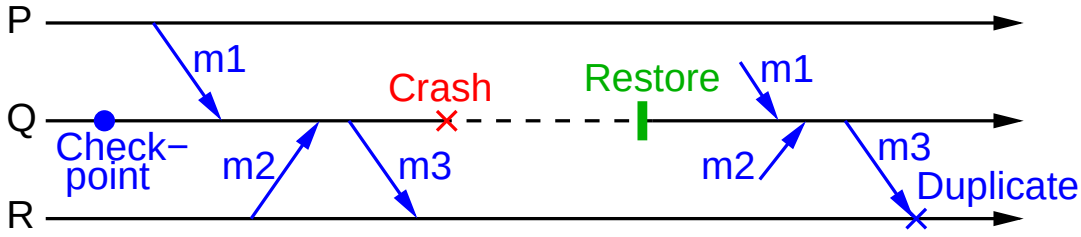


Notes for slide 316:

- Regarding forward and backward error recovery, respectively:
  Example: Reliable communication in computer networks: The retransmission of a faulty frame is a backward error recovery, because in the end one resets to the state in which the frame was not yet sent. The use of an error correcting code is a forward error recovery.
- Regarding the domino effect:
  If P2 crashes in the example, it can be reset to checkpoint 6 (CP6). However, the cut resulting from the current state of P1 and CP6 is not consistent (because of the last message). Therefore, P1 must also be reset (to CP5). However, the cut (CP5, CP6) is also not consistent (because of the penultimate message). Therefore an earlier checkpoint must be used for P2 (CP4). The cut (CP5, CP4) is also inconsistent, so that the reset continues until the initial state is finally reached.

11.4 Recovery ...

- Coordinated checkpoints
  - Chandy/Lamport algorithm (☞ **6.4**
  - alternatively: blocking 2 phase protocol
  - problem: requires to reset all processes
- Local checkpoints with message logging
  - goal: restore the crashed process to a state consistent with the current state of the other processes
  - reset to last checkpoint and restore the received messages



Notes for slide 317:

If Q crashes, it is reset to the checkpoint. The recorded messages m1 and m2 can then be replayed. Q then sends the message m3 again (assuming that the process behaves deterministically!). R recognizes this m3 as a duplicate and discards the message. After that the whole system is in a consistent state again.