Fault Tolerance

Introduction

Pedro F. Souto (pfs@fe.up.pt)

April 18, 2017

Fault Tolerance

Definition A system/component **fails** when it does not behave according to its specification.

Definition A system is **fault-tolerant** if it behaves correctly despite the failure of some of its components

- Obviously, no system tolerates the failure of **all** its components
- Usually, a system tolerates only some kinds of failures, as long as they do not occur too frequently or they only occur on some of its components

Observation Fault tolerance is achieved by design. We need to include some redundancy in the system:

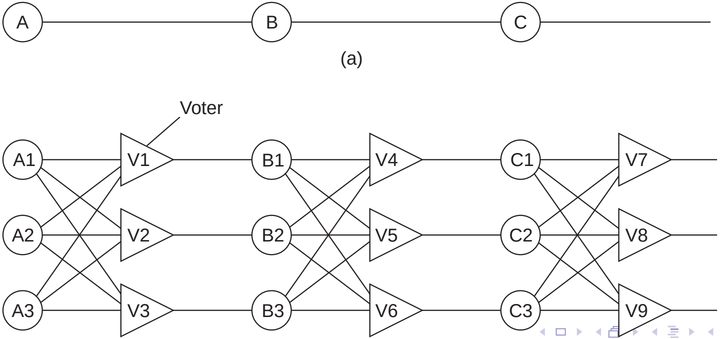
HW Processors, memory, I/O devices, communication links, ...

Time For executing additional tasks, e.g. retransmission of a packet

SW To manage the redundant HW, or the repetition of task or even n-version programming

Triple Modular Redundancy (TMR)

- Most commonly used HW-based FT-technique, originally proposed by von Neumman
- Each node is triplicated and works in parallel
- The output of each module is connected to a voting element, also triplicated, whose output is the majority of its inputs
- The configuration can be applied to each stage of a chain
  - It masks the occurence of one failure in each stage
  - What if a voter fails?



FT and Distributed Systems

Obs. Unless a distributed system is fault tolerant it will be less **reliable** than a non-distributed system

- A distributed system comprises more components than a non-distributed system
- In the 1980's, Lamport famously wrote in an email message:

  A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Obs. The inherent HW redundancy in a distributed system makes it particularly suitable for making it fault tolerant

- But, fault-tolerance does not emerge directly from distribution, it must be engineered

Reliability and Availability

Reliability ($R(t)$) the probability that a system **has not failed until** time t

- Particularly important for **mission**-oriented systems, such as spacecrafts, aircrafts or cars
- It is often characterized by the **mean time to failure (MTTF)**

Availability Assumes that a system may be repaired after failing.

Limiting the probability that a system is working correctly:
$\alpha = \frac{MTTF}{MTTF+MTTR}$, where *MTTR* is the mean time to repair

- Particularly important for systems like utilities, services on the web, that tolerate the occurrence of failures

Obs. Reliability and availability are somewhat orthogonal:

- A system A may be more reliable than system B and still be less available than system B
- A system A may be more available than system B and still be less reliable than system B

Failure Models (1/2)

Characterize a system in terms of the failures of its components, i.e. the deviations from their specified behavior

Crash a component behaves correctly until some time instant, after which it does not respond to any input

Omission a component does not respond to some of its inputs

- Loss of a message can be seen as an omission failure of the communication channel or of either processes at the channel ends

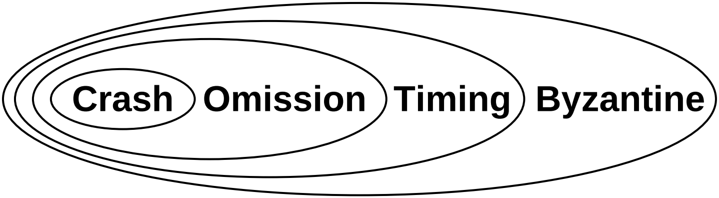
Timing/Performance a component does not respond on time, e.g. it may respond too early or too late

- Makes sense only on synchronous systems. Why?

Byzantine/arbitrary a component behaves in a totally arbitrary way

- For example, a process may send a message as if it were another process

Failure Models: Taxonomy (2/2)



Recovery Fault models also need to specify whether a faulty process may recover a **finite** number of times

- The practice is that if nothing is stated, then **they do not**

Failure and Synchrony Models

- The byzantine model is similar to the asynchronous model in that:
  - Neither model makes any assumption wrt the aspect of behavior it is supposed to describe
- In the absence of faults, the synchronous and the asynchronous models are **equivalent**
  - They can solve the same set of problems

Further Reading

- Section 8.1: Introduction to Fault Tolerance, Tanenbaum and van Steen, *Distributed Systems*, 2nd Ed.