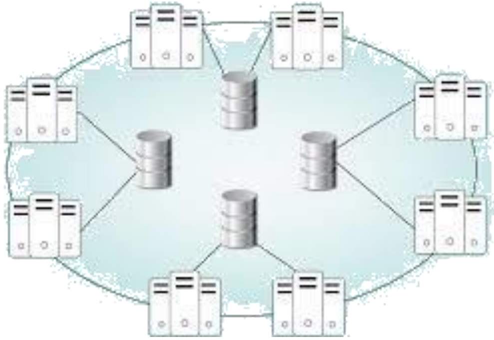
Distributed and Parallel Computer Systems

CSC 423

Spring 2021-2022

Lecture 12



Distributed Systems' Processes-2

INSTRUCTOR

DR / AYMAN SOLIMAN

➢ Contents

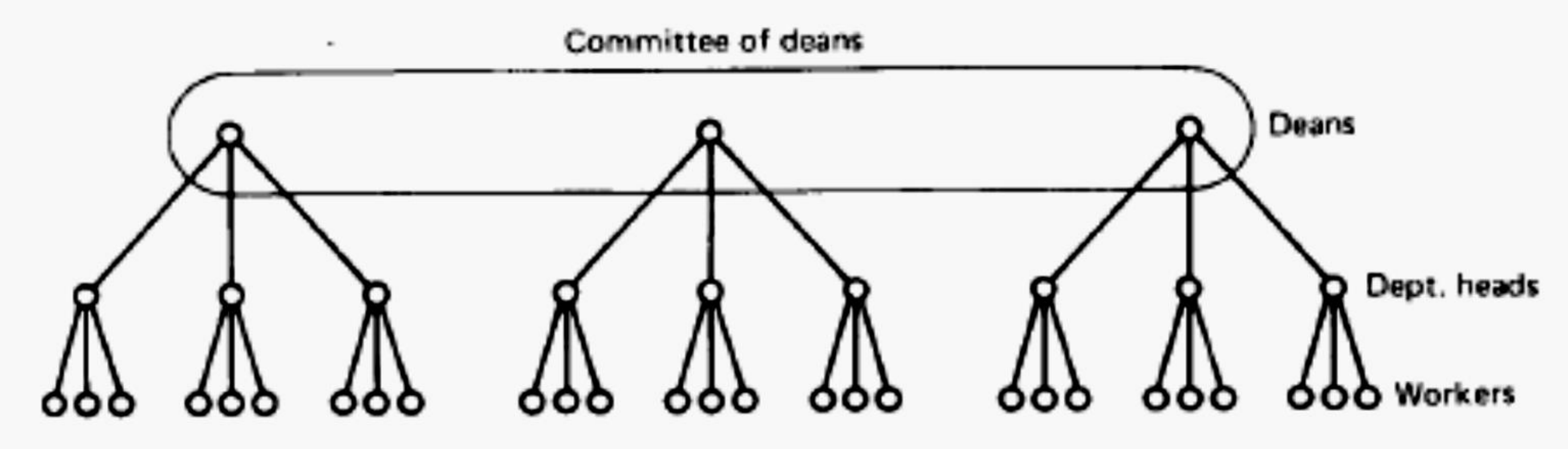
❑ Penalty points

- When a workstation owner is running processes on other people's machines, it accumulates penalty points, a fixed number per second. These points are added to its usage table entry.

- Usage table entries can be positive, zero, or negative.
  - A positive score indicates that the workstation is a net user of system resources,
  - A negative score means that it needs resources.
  - A zero score is neutral.

❑ Hierarchical Algorithm

- Centralized algorithms, such as up-down, do not scale well to large systems. The central node soon becomes a bottleneck, not to mention a single point of failure.
- This approach organizes the machines like people in corporate, military, academic, and other real-world hierarchies.
  - Some of the machines are workers and others are managers



Sender-Initiated Distributed Heuristic Algorithm

- When a process is created, the machine on which it originates sends probe messages to a randomly-chosen machine, asking if its load is below some threshold value. If so, the process is sent there.

- it should be observed that under conditions of heavy load, all machines will constantly send probes to other machines in a futile attempt to find one that is willing to accept more work.

Receiver-Initiated Distributed Heuristic Algorithm

- Algorithm is one initiated by an underloaded receiver.
- whenever a process finishes, the system checks to see if it has enough work. If not, it **picks some machine at random** and asks it for work.
- An advantage of this algorithm is that it does not put extra load on the system at critical times.

Bidding Algorithm

- The key players in the economy are the processes, which must buy CPU time to get their work done, and processors, which auction their cycles off to the highest bidder.
- Each processor advertises its approximate price by putting it in a publicly readable file.

SCHEDULING IN DISTRIBUTED SYSTEMS

- Each processor does its **own local scheduling** (assuming that it has multiple processes running on it), without regard to what the other processors are doing.

- When a **group of related**, heavily interacting processes are all running on different processors, independent scheduling is not always the most efficient way.

- The **basic difficulty** can be illustrated by an example in which processes *A* and *B* run on one processor and processes *C* and *D* run on another.

SCHEDULING IN DISTRIBUTED SYSTEMS

Time slot	Processor 0	Processor 1
0	A	C
1	B	D
2	A	C
3	B	D
4	A	C
5	B	D

(a)

Time slot	Processor 0	1	2	3	4	5	6	7
0	X				X			
1			X			X		
2		X			X		X	
3	X					X		
4		X		X				X
5			X		X			

(b)

- Several algorithms based on a concept he calls **co-scheduling**, which takes interprocess communication patterns into account while **scheduling to ensure** that all members of a group **run** at the same time.

- The **first algorithm** uses a **conceptual matrix** in which each column is the process table for one processor,

FAULT TOLERANCE

- A system is said to fail when it does not meet its specification.

Component Faults

- **Computer systems** can fail due to a fault in some component, such as a ***processor, memory, I/O device, cable, or software***.

FAULT TOLERANCE

- Faults are generally classified as transient, intermittent, or permanent.
  - **Transient faults** occur once and then disappear.
  - **An intermittent fault** occurs, then vanishes, then reappears, and so on.
  - A **permanent fault** is one that continues to exist until the faulty component is repaired.
- The goal of designing and building fault-tolerant systems is to ensure that the *system as a whole continues to function correctly, even in the presence of faults*.

System Failures

- In a critical distributed system, we are interested in making the system be able to survive component (in particular, processor) without faults.

- **Two types of processor faults can be distinguished:**

1. Fail-silent faults.

Faulty processor just **stops and does not respond** to subsequent input or produce further output

2. Byzantine faults.

Faulty processor **continues to run**, **issuing wrong** answers to questions,

Synchronous Vs Asynchronous Systems

- If one processor sends a message to another, it is guaranteed to get a reply within a time *T* known in advance.
- Failure to get a reply means that the receiving system has crashed.

Synchronous Vs Asynchronous Systems

- System that has the property of always responding to a message within a known finite bound if it is working is said to be **synchronous**.

- A system not having this property is said to be **asynchronous**.

- Asynchronous systems are going to be **harder to deal** with than synchronous ones.

Use of Redundancy

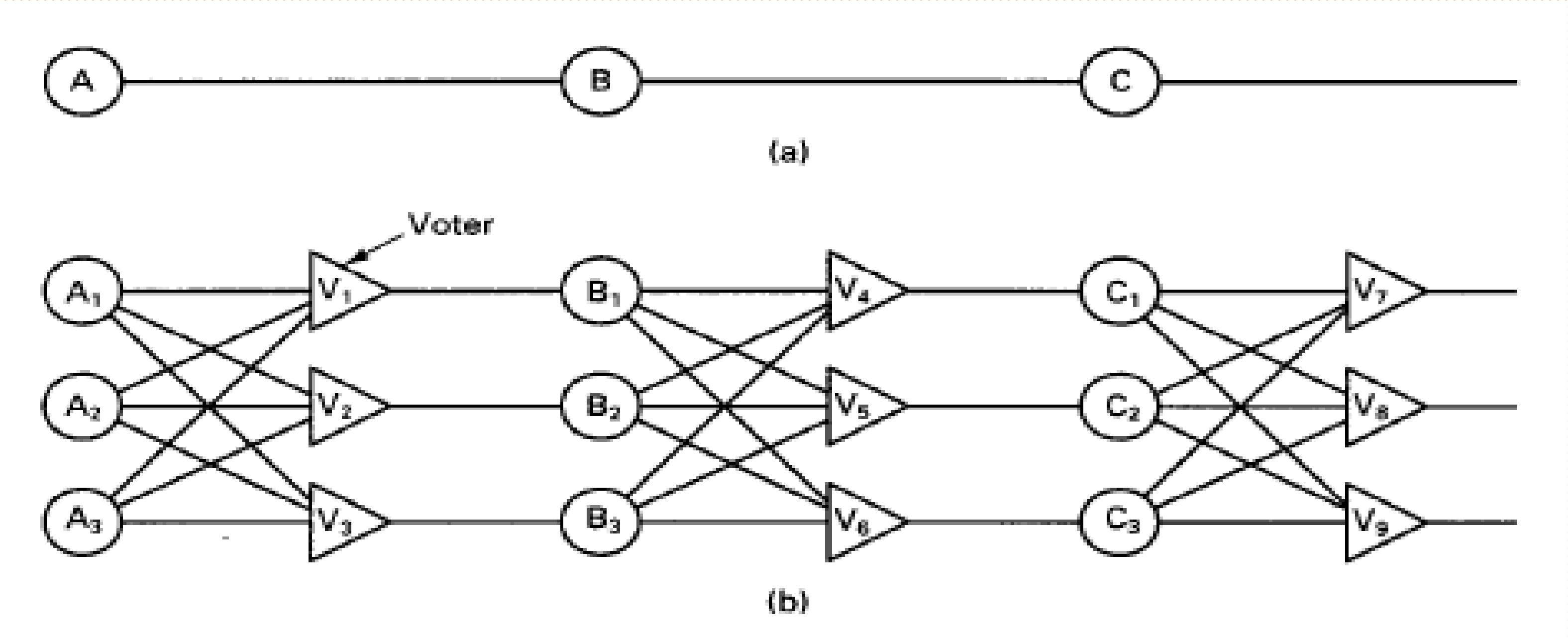
- The general approach to fault tolerance is to use redundancy
- Three kinds are possible:
  - **Information redundancy,**
    - Extra bits are added to allow recovery from garbled bits.
  - **Time redundancy,**
    - *an action is performed, and then, if need be, it is performed again.*
    - Time redundancy is especially helpful when the faults are **transient or intermittent**.
  - **Physical redundancy**.
    - **extra equipment is added** to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components (**permanent fault**)

Fault Tolerance Using Active Replication

- **Active replication** is a well-known technique for providing fault tolerance using physical redundancy.

  - It is used in biology (mammals have two eyes, two ears, etc.),

  - If all three inputs are different, the output is undefined. This kind of design is known as TMR (Triple Modular Redundancy).



Triple modular redundancy

Fault Tolerance Using Primary Backup

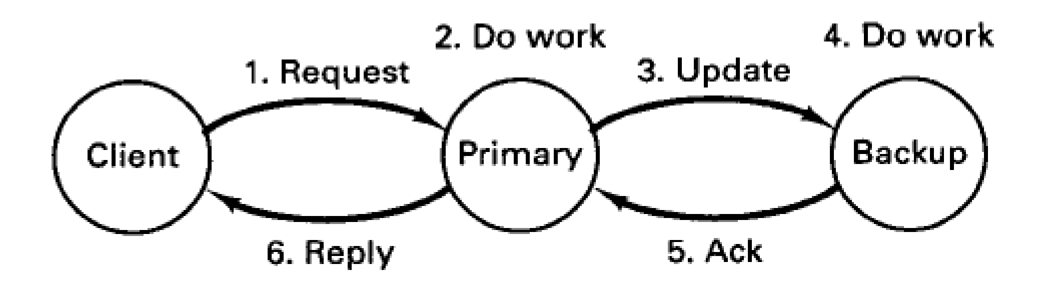
- The essential idea of the **primary-backup method** is that at any one instant, one server is the primary and does all the work. If the primary fails, the backup takes over.

Fault Tolerance Using Primary Backup

- Primary-backup fault tolerance has two major **advantages over active replication**.
  - First, it is simpler during normal operation since messages go to just one server (the primary) and not to a whole group.
    - The problems associated with ordering these **messages** also disappear.
  - Second, in practice it requires fewer machines, because at any instant one primary and one backup is needed

Fault Tolerance Using Primary Backup

A simple primary-backup protocol on a write operation.



Agreement in Faulty Systems

- The general goal of distributed agreement algorithms is to have all the non-faulty processors ***reach consensus*** on some issue, and do that within a finite number of steps.

  - Examples are electing a coordinator, deciding whether to commit a transaction or not, dividing up tasks among workers, synchronization, and so on.

Agreement in Faulty Systems

- Different cases are possible depending on system parameters, including:

  1. Are messages delivered reliably all the time?

  2. Can processes crash?
     - if so, fail-silent or Byzantine

  3. Is the system synchronous or asynchronous?

Agreement in Faulty Systems

- Let us look at the "easy" case of perfect processors but **communication lines that can lose messages**. There is a famous problem, known as the **two-army problem.**

  - **two-army problem**

  - **the sender of the last message does not know if the last message arrived.**

  - **Even with nonfaulty processors (generals), agreement between even two processes is not possible in the face of unreliable communication.**

Agreement in Faulty Systems

- Now let us assume that the communication is perfect but the processors are not.

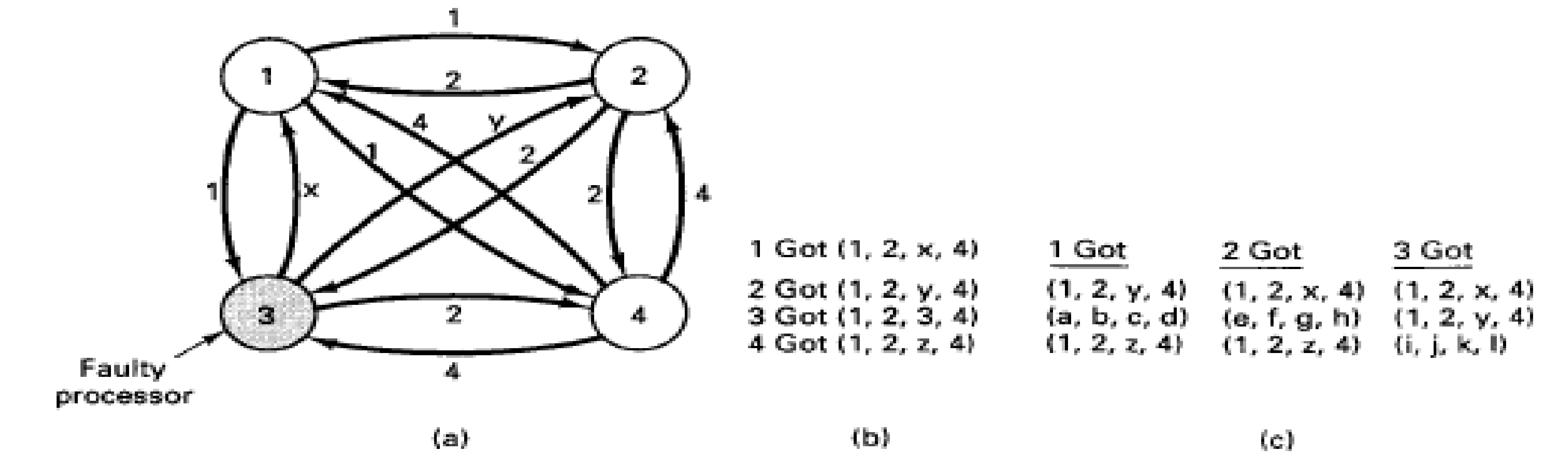
- The classical problem occurs in a military setting and is called the **Byzantine generals problem.**

- The goal of the problem is for the generals to exchange troop strengths, so that at the end of the algorithm, each general has a vector of length *n* corresponding to all the armies.

Agreement in Faulty Systems

- If general / is loyal, then element / is his troop strength; otherwise, it is undefined.
- A recursive algorithm solves this problem under certain conditions.
- we illustrate the working of the algorithm for the case of $n = 4$ and $m = 1$ For these parameters, the algorithm operates in four steps.

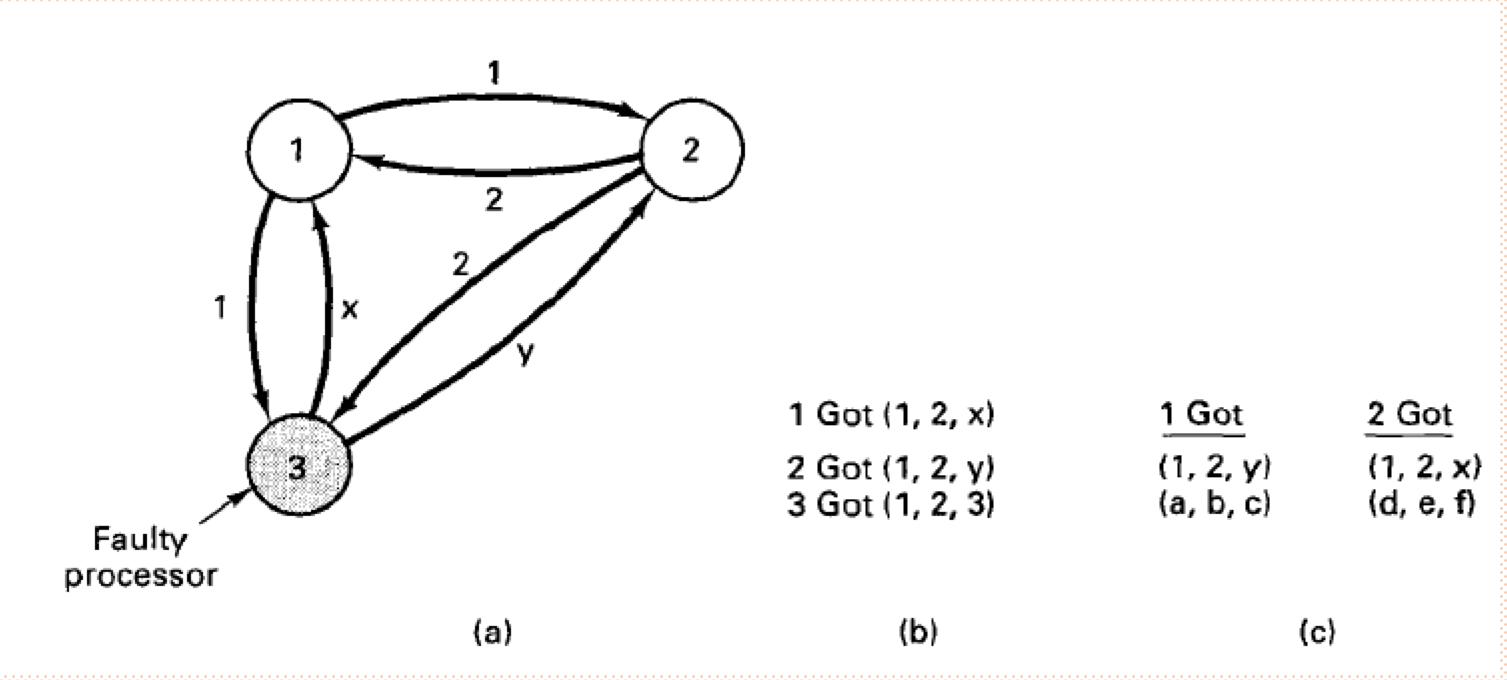
Agreement in Faulty Systems



- in step 4, each general examines the i^{th} element of each of the newly received vectors. If any value has a majority, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked unknown. From Fig. (c) we see that generals 1, 2, and 4 all come to agreement on

(1, 2, UNKNOWN, 4)

Agreement in Faulty Systems



Agreement in Faulty Systems

- Lamport et al. (1982) proved that in a system with m faulty processors, agreement can be achieved only if 2m + 1 correctly functioning processors are present


Thank you