Introduction

- **Def**: A distributed system is a collection of independent computers that appears to its users as a single coherent system. In which hardware and software components located at network computers communicate and coordinate their actions using only messages.
- In distributed systems the computer power is distributed.
- The parallel systems are able to do calculations in parallel.
- In parallel computing, all processors may have access to a shared memory to exchange information between processors.
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.

Types of memory

- **Shared memory**:
  - From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory
  - can be used in parallel systems. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.
- **Distributed memory**:
  - In hardware, refers to network based memory access for physical memory that is not common.
  - As a programming model, tasks can only logically "see" local machine memory
- **Distributed shared memory**: it is possible to access to other physical memory using an Ethernet bus for example
- Distributed system cannot be in the same hardware

Characteristics

- **Concurrency**: is focus on cyberphysical systems. Concurrency is a property of a system representing the fact that multiple activities are executed at the same time. Only executing two different tasks simultaneously yields true parallelism.
- Shared resources (memory, bus)
- No global clock: no time stamp, it is an issue
- Independent failure; some parts may fail due to power consumption
- Heterogeneity: different types of devices working together. Need some type of virtualization

- Openness: Be able to interact with services from other open systems, irrespective of the underlying environment.
- Scalability: a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added

Distributed System Categories

- Distributed computer system:
  - Cluster and Grid computer systems for example
  - High-speed network (for security reasons) is not connect to Master node.
  - High performance computation
- Distributed Information systems:
  - Client-Server applications. Transaction Processing Systems for example
  - Not only one type of network
  - Paradigm: not try to implement high performance computation in this form, information systems are meant to work with databases or to produce information.
  - A computer information system is a system composed of people and computers that processes or interprets information. The term is also sometimes used in more restricted senses to refer to only the software used to run a computerized database or to refer to only a computer system.
- Distributed Pervasive systems:
  - Electronic Health Care Systems for example
  - Store and process information
  - Small nodes, mobile, usually embedded, battery-powered
  - No calculation is done in the device, all are done in an external device
  - computing is made to appear anytime and everywhere.

Advantages

- Efficiency: more processing for less effort
- Concurrency: some computation can be done in parallel
- Reliability: "backup device"
- Scalability: easily extended

Disadvantages

- Complexity:
  - Concurrency is sometimes hard to achieve

  - Reliability needs to be implemented
  - Scalability
- Security
- Manageability:
  - The key to high availability is redundancy: if you have more than one of everything, you're not susceptible to any single points of failure
  - Manageability includes both the ability to detect faults (or the absence of faults) and the ability to automatically react to the

Architecture

- Architecture usually serves some purpose
  - Ensure that the structure will meet present and likely future demands
  - Reliable
  - Manageable: manage resources, logistics
  - Adaptable
  - Cost-Effective
- Architectural elements in developing distributed embedded systems
  - Communication entities: clusters, micro controllers, memory for examples
  - Communication paradigms: MPI
  - Roles and responsibilities: several clients will access the back-end. Roles implies access restrictions
  - Placement: distribute tasks, data. Virtualization
- Should be stable

Viewpoints

- Different diagrams to capture structure and behavior (UML)
- Different roles have different viewpoints on systems

Kruchten's 4+1

- Logical view:
  - End-user functionality
  - relation between classes.
  - Consider extend software
- Process view:
  - communication between objects.
  - Help with planning communication overload (threads, tasks and their interdependencies)
- Physical view:

  - System engineer viewpoint
  - Describes the mapping from software to hardware
  - Structural Information: Processing units, memories, networks, data, communication
- Development view:
  - Software engineer viewpoint
  - Describes the organization of software
  - Modules and dependencies
- Use case View:
  - End users and developers view
  - Describes user focused scenarios

Resource management protocols (NO SLIDES)

Priority inversion

- Resource locking prevents high priority tasks to access the resource. So a high priority task is indirectly preempted by a lower priority task effectively "inverting" the relative priorities of the two tasks.
- Priority inversion: critical section access is not in priority order
- Solution:
  - disabling interruption
  - priority ceiling: CS manager has highest priority
  - priority inheritance: preemption prevention via temporarily high priority assignment for CS release
  - random boosting: ready tasks holding locks are randomly boosted in priority until they exit the critical section
- Deletion safety: process holding a Mutex cannot be accidentally deleted, this also prevents deadlocks

Protocols

- Non Preemption Protocol - NPP: any task that locks a resource receive the highest priority of among all the task
- Highest Lower Protocol - HLP: any task that locks a resource receive the highest priority of among all the running tasks
- Priority Inheritance Protocol - PIP: assign the highest priority only another task wants to use the resource
- Priority Ceiling Protocol - PCP: Enter to CS only if it is free and there is no chained block risk (a HP task is waiting for a resource locked by a lower priority task that is waiting for a resource locked by even a lower priority task)

Multiprocessor

- Remote blocking: A resource is locked by a task on a different processor
- Needs an extension of the protocols (more info slides)

Time and clocks

- Use to causally order events (cause & effect)
- Ensure temporal correctness
- Determine deadlines and temporal coherences
- Embedded system development: debugging, profiling and trace
- Synchronization
- Coordination
- Event serialization
- Causality:
  - $a \mapsto b$ means that *b is causally effect by a*, also implies that *a happened before b*
  - $a \succ b$ means that *a happened before b* (temporal order)

Ordering mechanism

- Central observer: (monitor)
  - Are simple architectures
  - Slow down the system, and non-functional behavior
- Descentralized synchronization
  - Logical clocks
    - provide causally order
  - Real/ physical clocks
    - provide temporal order
    - Differ in resolution, accuracy and drift
    - it is possible to calculate elapsed time, current time

Timestamps

- can be compared to each other with respect to their temporal precedence, if and only if they were derived from the same global clock or a set of synchronized clocks.
- need information about what clock is used, resolution, clock drift
- **clock drifting**: Clock drift refers to several related phenomena where a clock does not run at exactly the same rate as a reference clock. That is, after some time the clock "drifts apart" or gradually desynchronizes from the other clock.

Physical clocks

- Physical clock is a device which is based on some periodic physical oscillation (e.g. pendulum, quartz crystal, atom) in order to measure the progression of time.
- Have desviations from the ideal clock
- Granularity: duration between two consecutive ticks
- Clock measure $\Delta t_g = n_g/f$, n_g is the numbers of ticks
- Drift: $d = \frac{n_g}{f.\Delta t_g}$, ideally 1
- Deviation: $\delta = \|1 - d\|$

Clock synchronization

- Two clocks hardly ever run synchronously
- Drift: rate deviation (per tick)
- Skew: deviation at specific point in time
- Apply adjustments continuously (e.g. Linux: POSIX, adjtime an system calls)

Christian's algorithm

- synchronize with other machine
- Client gets data from a time server (e.g. access to radio clock)
- Computation of round trip time to compensate delay while adjusting own clock:
- With T0 = request sent time ant T1 = receive time; assuming network delays symmetric $T_new = T_server + (T1 - T0)/2$

Berkeley algorithm

- Assumption: no machine has an accurate time source
- There is a master clock
- Having multiple computers in a network, drift tendencies can be minimized by calculating average
- *outlayer* clocks are not considered in the estimation

Network time protocols

- Use Stratums to clasify clocks.
  - Stratum 0: high precision clocks/ reference clocks
  - Stratum 16: unsynchronized
- Intended for multiple up- & downstream servers

- dispersion, jitter, clock filter calculation
- protocol definitions for declaring falsetickers

Simple network protocol

- No mitigation algorithms / peer process operations / clock filter algorithm (compared with NTP)
- same protocol but time synchronization quality lower
- recommended to use only at highest stratums
- intended for primary servers with single reference clock; no upstream servers

Precision time protocol

- master slave architecture: ordinary clock and boundary
- To transfer clock to another LAN
- low jitter, low latency

Logical clocks

- give us information about which event happens before, after or in concurrency.

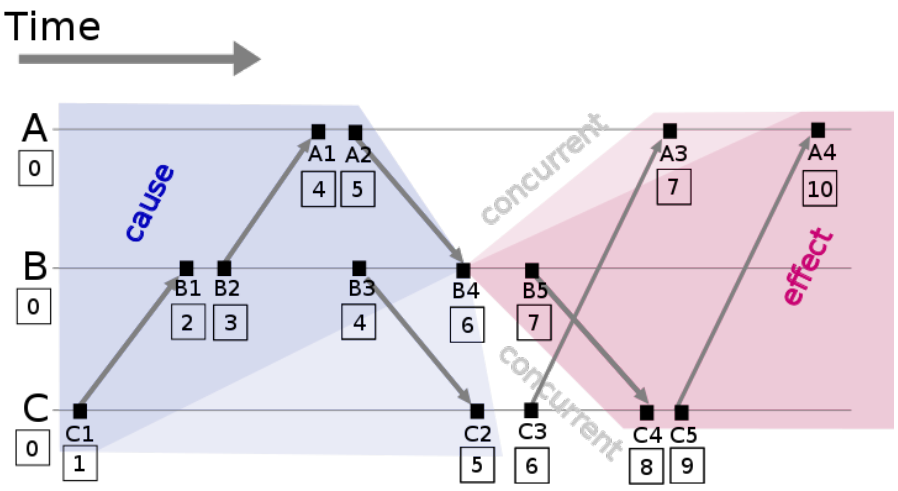
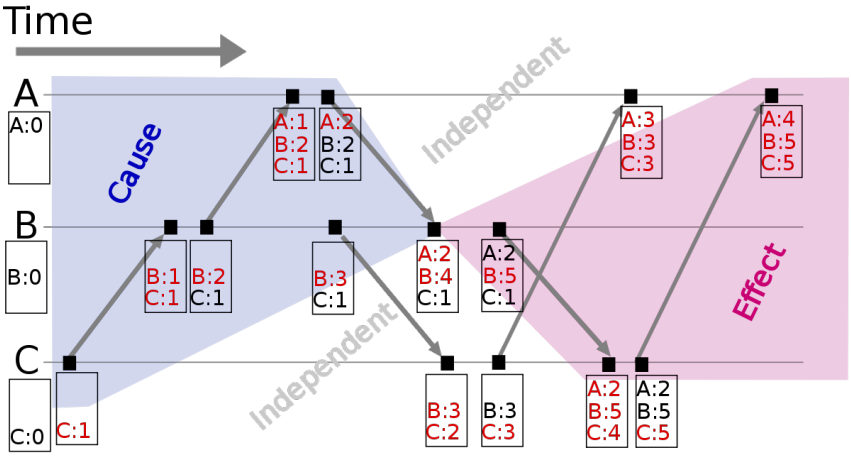


Figure 1: Lamport's clock

- Lamport's clock: every Process P keeps scalar counter C, increments it at each event, sends it with each send event
- Vector clock: B4 and C2 are concurrent $B4 \| B2$ since $4 \geq 3$ and $1 \leq 2$



Matrix clock

- Idea: keeping local state (LC), global state (VC) and each node's view of the state (MC) a n x n matrix (n= number of nodes)
- "I know that you know about me and the others"

CAP theorem

- tells you on what to focus your system

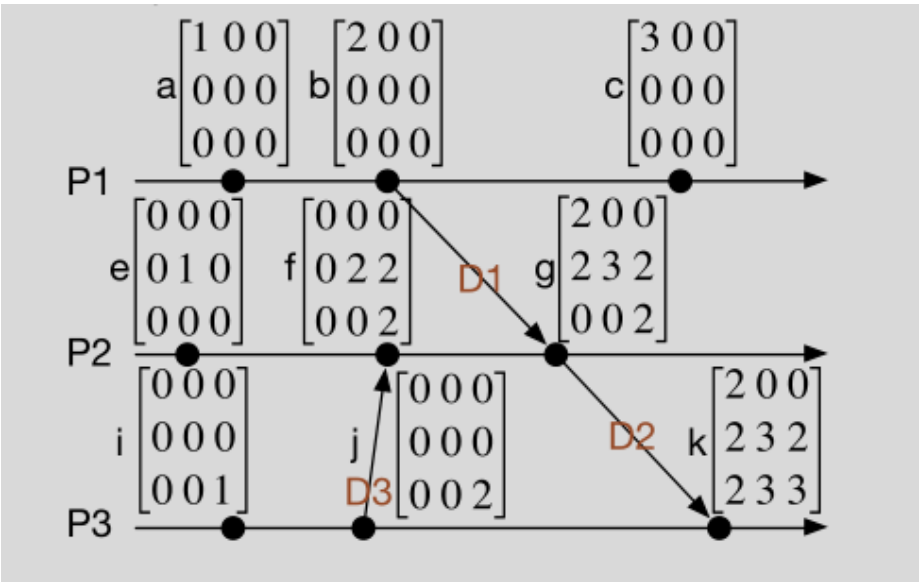


Figure 2: Matrix clock