Sistemas Distribuídos

Introductory Concepts

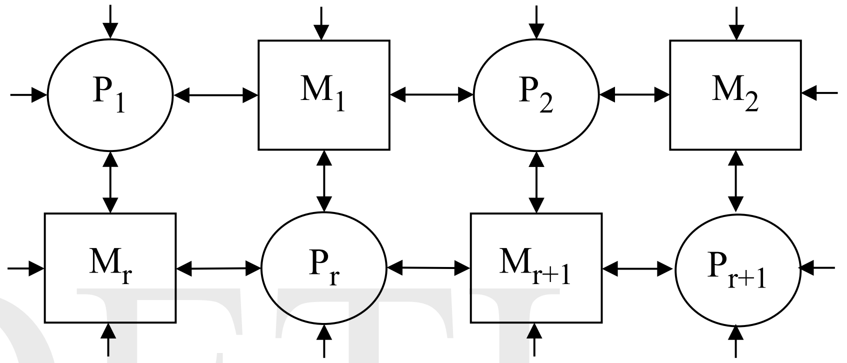
António Rui Borges

Summary

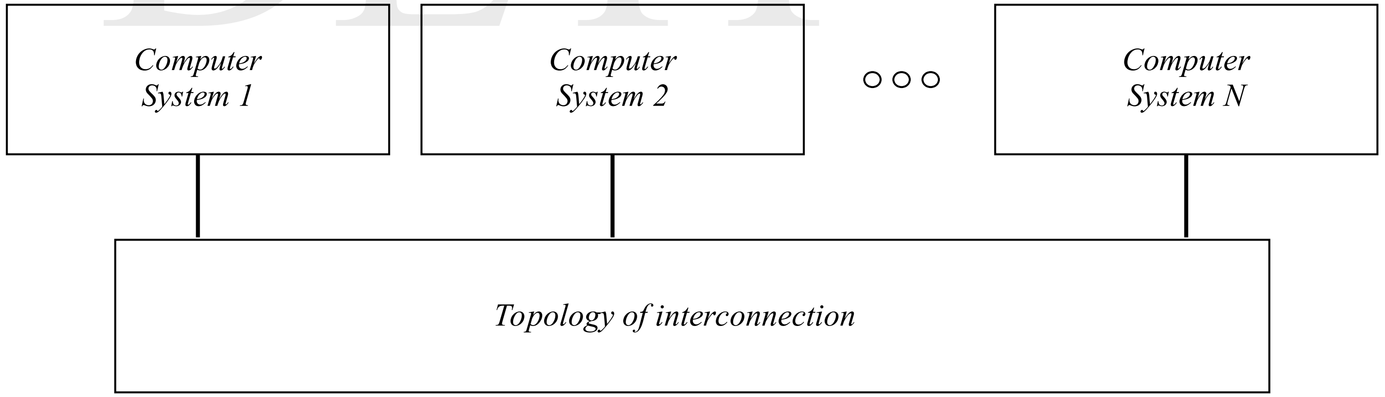
- *Parallel computer systems*
- *Characterization of distributed systems*
- *Issues to be taken into account in the project of distributed systems*
  - *Heterogeneity*
  - *Transparency*
  - *Openness*
  - *Security of the information flows*
  - *Scalability*
  - *Fault handling*
  - *Concurrency*
- *Ubiquitous computing* (*Internet Of Things*)
- *Cloud computing*
- *Suggested reading*

Parallel computer systems

Mesh multiprocessor



Computer network



Distributed systems - 1

The conventional definition of ***distributed system*** states that one is dealing with *an* [*operating*] *system* whose *components, located in the different processing nodes of a parallel computer system, communicate and coordinate their actions through message passing* [Coulouris et als.].

The main motivation underlying the construction and the use of *distributed systems* is resource sharing. *Resource* is to be understood here as an abstract entity which embodies something *material*, like processors, network infrastructures, storage devices and more or less specialized peripherals, or *immaterial*, like information taken in the most wide sense, or the functionalities that operate upon it.

This sharing is translated in two distinct ways

- *application parallelization* – by taking advantage of the multiple processors and other *hardware* components of the parallel computer system, one tries to ensure a faster and more efficient program execution
- *service availability* – by managing a set of somewhat related resources, one tries to organize them in such a way that an uniform communication interface based on a well-defined API may be provided.

Distributed systems - 2

An alternative definition of ***distributed system***, although essentially equivalent, is to say that one is dealing with *a collection of independent computing systems perceived by the users as a coherent system* [Tanenbaum et als.].

Several consequences stem from it

- *parallelism* – the existence of independent computer systems gives rise to simultaneous and autonomous threads of execution
- *global internal state* – the need to provide a coherent working image requires some form of activity coordination among the different processing nodes
- *communication through message passing* – the fact that no assumptions are made about how the computer systems are interconnected, entails a minimalist communication mechanism based on message passing
- *scalability* – the faculty of expanding the system by integration of more processing nodes is an immediate outcome of the prescribed organization
- *failure handling* – any of the components of the distributed system may fail at any time, leaving the remaining components in operation.

Types of distributed systems - 1

The way *distributed systems* are organized is strictly dependent on the kind of problems they are aimed to solve.

Four types of distributed systems are among the most common in practice

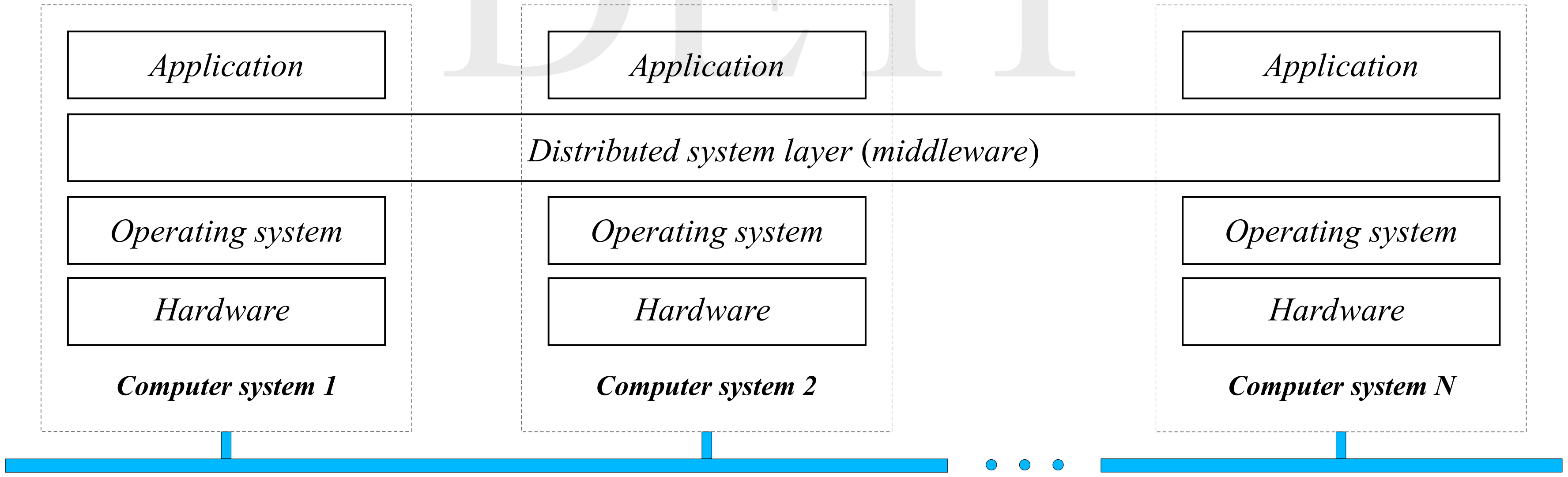
- *cluster systems* – they consist typically of a collection of computer systems interconnected by a high speed network; the aim is application parallelization; typically, the computer systems are identical, all run the same operating system and share the same network; there is a single master node that handles the allocation of processing nodes to a particular program, maintains a queue of submitted jobs and interfaces with the system users
- *grid systems* – no assumptions are made here concerning the hardware platforms which form the processing nodes, neither the operating systems they run, nor the interconnecting networks; the key issue is bringing together the resources from different organizations so that a community of people may cooperate; a virtual organization is thus formed, where only its members have access rights to the resources provided

Types of distributed systems - 2

- *transaction information systems* – the main processing nodes run applications whose primary goal is to manage information mostly organized in databases, called *servers*; this information is made available to remote-run programs, the *clients*, through a succession of operations of the type request / reply; integration at the lowest level allow the clients to wrap together a group of requests, possibly to different servers, into a single larger request and have it executed as a *transaction*, that is, all or none of the sub-requests are executed
- *enterprise integration systems* – as the applications become decoupled from the associated databases, it turns more and more apparent that means are needed for application integration independent of the databases; in particular, application components should be able to communicate directly with one another and not merely through the request / reply behavior supported by transaction information systems.

Distributed systems - 3

Assuming generic computer systems, as processing nodes, and diverse communication networks, as interconnection infrastructures, while simultaneously keeping an integrated view of its functionality, *distributed systems* can be understood as a software layer, located between the applications and the local operating systems, aiming to create a model of programmable virtual machine that hides all the underlying heterogeneity – the so called *middleware*.



Design of distributed systems / applications

The design of *distributed systems*, as well as the development applications run in them, requires that several issues whose importance is crucial to good performance to be considered.

The following are singled out

- the [possible] *heterogeneity* of the components that form the parallel computer system
- its *transparency*
- the degree of *openness*
- the *security* of the exchanged information flows and of the stored information
- the potential it presents for *scalability*
- the *failure handling*
- its potential for *concurrency*.

Heterogeneity - 1

The *heterogeneity* of a parallel computer system becomes apparent in multiple forms

- the interconnection of different types of computer networks
- the use of different hardware platforms as processing nodes
- different operating systems running simultaneously in different nodes
- different programming languages used in the coding of different software components of the same application
- the interconnection of functionalities provided by different suppliers.

In view of such diverse reality, it is mandatory to find solutions which provide a coherent approach.

Heterogeneity - 2

The solutions should be *localized*, introducing a greater or lesser degree of uniformity at a specific level of the communication chain

- specification and implementation of a network architecture
- conversion the data representation specificities presented by each processor into a common format
- harmonization of the *communication programming interface* offered by the different operating systems
- adjustment of the data structures implemented by different programming languages into a common format
- development of standards accepted and complied by different manufacturers.

Transparency - 1

The degree of *transparency* of a distributed system is a feature that expresses the greater or lesser success in masking the underlying complexity. Thus, making the operation of a distributed system *transparent* means describing its functionality in an integrated fashion, conceptually simple, instead of resulting from the interaction of a collection of independent components.

The *transparency* goal is, therefore, to hide the resources which are not directly relevant to the task being carried out, making them anonymous both to the user and/or the applications programmer.

The degree of *transparency* with which the different resources may be accessed and operated, portrays immediately the degree of abstraction and operativeness of the *middleware* layer.

Transparency - 2

Transparency modes

- *access transparency* – when the same operations are performed to access local and remote resources
- *position transparency* – when resource access is carried out without the knowledge of its precise physical or network location
- *network transparency* – when access and position transparency exist concomitantly
- *mobility transparency* – when the client access location to the resources may change within the system without affecting the operation being performed
- *migration transparency* – when the resources may be moved without affecting their access
- *relocation transparency* – when the resources may be moved without affecting their access, even when an access is taking place (strong version of *migration transparency*)

Transparency - 3

- *replication transparency* – when it is possible to instantiate multiple copies of the same resource without the fact of doing so becoming obvious
- *performance transparency* – when a dynamic system reconfiguration may occur to cope with load variations
- *scaling transparency* – when the system and applications may expand in scale without requiring any change in the system structure and on the application algorithms
- *concurrency transparency* – when access to shared resources is performed in parallel by multiple entities without being aware of one another (data should always remain consistent)
- *failure transparency* – when failures, occurring in the hardware and/or software system components, can be masked and, therefore, the tasks in execution be terminated.

Openness

The degree of *openness* of a computer system is a feature which determines its greater or lesser capability to be extended and re-implemented in different ways.

For *distributed systems*, *openness* is fundamentally related to the capability of incorporating new services and of making them available to a wide variety of applications.

It is, therefore, necessary

- to establish an uniform communication mechanism on access to shared resources
- to publish the main APIs
- to ensure a strict conformity, at the design and implementation levels, of each new component with the relevant standard.

Security of information - 1

The generalized service availability and the appetence demonstrated by parallel computer systems for the cooperative execution of applications raise the question of how safe are the exchanged information flows and of how safe is the data stored in the associated resources.

The *security of information* may be envisaged at three levels

- *confidentiality* – protection against its disclose to non-authorized entities
- *integrity* – protection against it modification or its corruption
- *availability* – protection against interference that disturb the access channels.

Security of information - 2

Some measures may be taken to reduce the risks

- *introduction of firewalls* – by creating a barrier around a local network which limits the input and output traffic to well-defined channels
- *message encryption* – by scrambling the contents of the information flows
- *use of electronic signatures* – for certification of the message author.

There are, however, situations of difficult solution

- *denial of service attacks* – when the service suppliers are flooded by a huge number of phony requests, which leads to service shutdown, preventing the access of the true users
- *security of mobile code* – executable files sent in attachment to messages, download of *plug-ins* and *applets*, for instance, may lead to undesirable system behavior, completely out of control of the local user.

Scalability

Distributed systems must allow *increases of scale*. That is, in view of a significant increase of the number and/or the size of the available resources and/or the number of the users which request them, must still keep an appropriate performance.

Some of the challenges one must face are

- *expansion of the physical resources* – increase of scale means in this sense that the amount of the required resources to service n users should not be larger than $O(n)$
- *performance losses* – increase of scale means in this sense that the performance loss resulting from the processing of a task should be about $O(log_2\ n)$ at the most, where n is some parameter measuring the task size
- *preventing future bottlenecks* – increase of scale means in this sense that an effective planning about the resources size and the access procedures must exist so that, taking into account the predictable future growth, prevents their exhaustion and the resulting performance losses.

Failure handling - 1

Distributed systems, being built out of multiple hardware and software components, are susceptible to partial failure of extreme varied sort. Dealing with it is, therefore, very difficult.

Some types of failure are easily detectable, but many others are masked by the complexity of the system. The great challenge is how to manage the system in an environment where some of the failures can not be detected, but whose existence one may merely suspect of.

Basic strategies for failure handling

- *masking the failures* – some of the detected failures can be hidden or, at least, their effect be made less severe
  - lost messages, if detected, can be transmitted again in many cases
  - resource replication can allow for the safeguarding of stored information when there is data corruption in some of the copies

Failure handling - 2

Basic strategies for failure handling (continuation)

- *recovering from failures* – in order to make it possible, it is necessary to design the software components in such a way that the internal states of the involved processes are periodically stored; when a failure occurs, processing is restarted from the last saved state

in practice, this may also require code migration to other hardware resources and the use of updated copies of the data

- *tolerating failures* – some failures have to be tolerated; any attempt to solve them is pointless or impractical; in these cases, instead of leaving the user stranded, while successive access attempts are carried out, it is better to let him/her know of the fact.

Concurrency

Being resource sharing the main motivation which leads to the design and the implementation of *distributed systems*, it is paramount to ensure that the entity, or entities, managing access to the shared resource, impose mutual exclusion for the operation to be performed in a consistent fashion.

When access control is *centralized*, that is, dependent of a single entity, there is a simple solution based on standard devices for imposing mutual exclusion and synchronization, developed for multiprogramming environments in monoprocessors.

When, however, access control to the resource is *distributed*, dependent of several independent entities, the solution is a lot more complex because one has first to tackle with the problem of how to synchronize distinct entities not subject to a global clock.

Ubiquitous computing (*Internet Of Things*)

Principles

- *to make the communication natural* – to provide intelligent interface devices among computer systems and human beings, or other computer systems, in order to make information exchange spontaneous
- *to make computing sensitive to context* – to select the actions to be taken which are specific to current scenery interpretation.

Important application areas

- *intelligent houses* (*domotics* or *house automation*) – setting up house appliances, remote control and monitoring, establishing surveillance procedures, etc
- *autonomous vehicles* – vehicle to vehicle communication, enforcing of scenery dependent traffic rules, etc.

Cloud computing

Principles

- *resources on demand* – making available to potential users different kinds of service
  - specific applications, run remotely (*software as a service*)
  - specific hardware means for computation (*platform as a service*)
  - specific resources, such as mass storage or tailored running environments (*infrastructure as a service*)
- *indirect access* – resources are accessed remotely through standard protocols run from local computer systems, tablets or smart phones
- *scalability* – the system should have the ability to adapt to users needs, creating the illusion that the resources are unlimited.

Suggested reading

- *Distributed Systems: Concepts and Design, 4th Edition, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Chapter 1: *Characterization of distributed systems*
- *Distributed Systems: Principles and Paradigms, 2nd Edition,* Tanenbaum, van Steen, Pearson Education Inc.
  - Chapter 1: *Introduction*