Distributed systems and middleware: patterns and frameworks

prof. Paolo Ciancarini

(University of Bologna, Italy)

Innopolis University, 2020

Who am I

Professor of Computer Science@UniBo

Researcher on sw technologies for distributed programming

… and who are you?

Goals and prerequisites

- This course presents the fundamental ideas on distributed systems and their principles of construction exploiting middleware and languages for distributed programming

- Prerequisites:
  - Be able to program (object oriented)
  - Basic notions of networking
  - Basic notions of operating systems
  - Basic notions of concurrent programming

Goals of this course

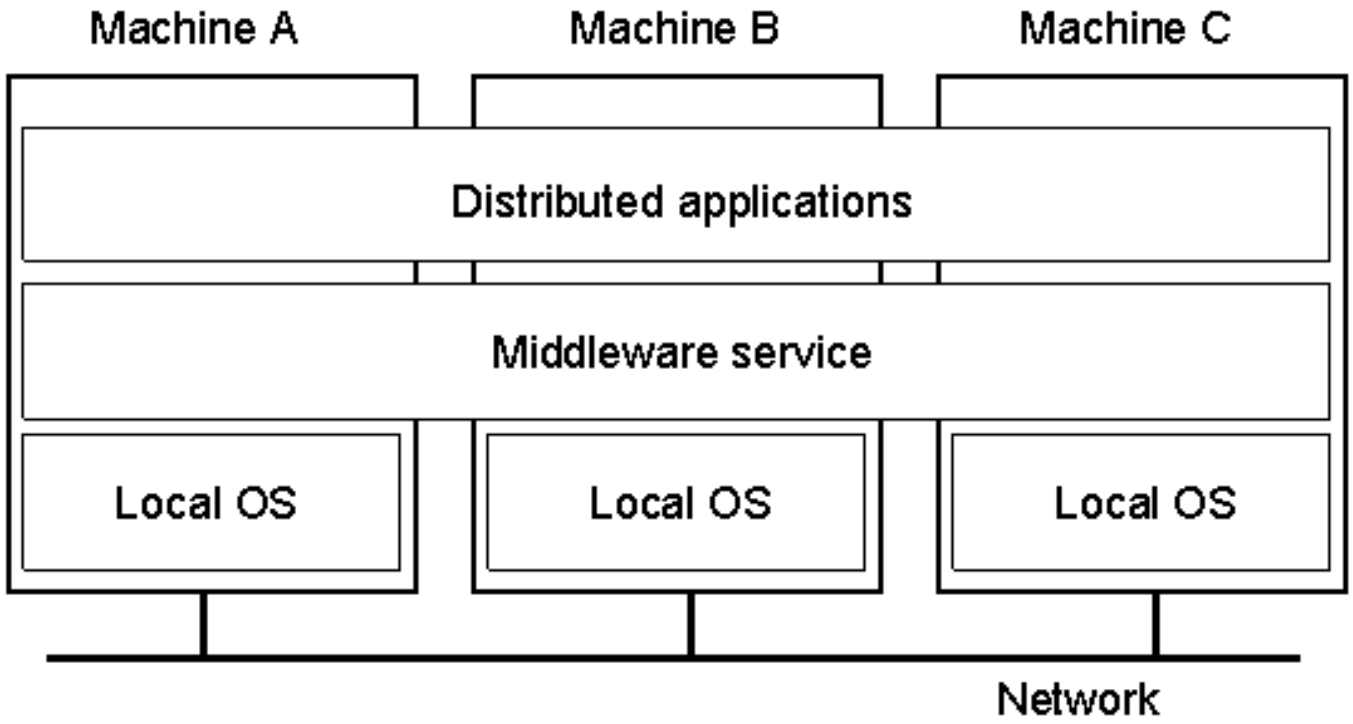
- Understand principles and concepts underlying **distributed systems and middleware:** communication, concurrency, coordination and related algorithms
- Learn how to **design** simple reliable **distributed applications**
- Gain **practical** experience on the **development** of simple distributed systems (eg. in **Java** or **Go**)
- Improve your software engineering expertise

Learning objectives

- Describe the non-functional characteristics of distributed applications
- Compare different types of middleware systems.
- Design, implement, and deploy distributed systems using the models of: web applications, web services, remote objects, and asynchronous messaging.
- Understand the challenge of managing time in a distributed system, and implement a means of assessing a distributed system's state.
- Understand transactions and implement a two phase commit protocol.
- Demonstrate deploying containers to cloud infrastructure
- Understand the problem of distributed consensus and design solutions
- Applications: distributed file systems, blockchains, microservices

Definition of a Distributed System

A distributed system is a collection of independent computers that appears to its users as a single coherent system



Leslie Lamport

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

Distributed vs centralized

Centralized system: state stored on a single computer

- simpler to design
- Easier to understand

Distributed system: state divided over multiple computers

- Scalable
- Robust
- Complex

Distributed systems: Why?

Share expensive resources

Distributed systems: Why?

Make information accessible

on large scale

with measurable Quality of Service

Distributed Sistems: Why?

Enable distant communication



Distributed systems: Why?

Enable financial transactions

Which problems?

Access (eg. How do we name resources?)

Transparency

Fault tolerance

Latency

… programming without shared memory and no absolute (unique) time

Transparency

- A transparency is some aspect of a distributed system that is hidden from the user (programmer, system developer, user or application program).
- A transparency is provided by including some set of mechanisms in the distributed system at a layer below the interface where the transparency is required.
- A number of basic transparencies have been defined for a distributed system; not all of these are appropriate for every system, or are available at the same level of interface.
- In fact, all transparencies have an associated cost, and it is extremely important for the distributed system implementor to be aware of this.

There are several transparencies

Transparency: access

Access transparency There should be no difference between local and remote access methods. In other words, the cost of explicit communication may be hidden. For instance, from a user's point of view, access to a remote service such as a printer should be identical with access to a local printer. From a programmers point of view, the access method to a remote object may be identical to access a local object of the same class.

Moreover, since the semantics of remote access are more complex, particularly in case of failure, this means that a local access should be a special case of remote access.
Remote access will not always look like local access in that certain facilities may not be reasonable to support (for example, global exhaustive searching of a distributed system for a single object may be unreasonable in terms of network traffic).

Transparency: concurrency

Concurrency transparency Users and Applications should be able to access shared data or objects without interference between each other.

This requires very complex mechanisms in a distributed system, since there exists true concurrency rather than the simulated concurrency typical of a centralized system.

For example, a distributed printing service must provide the same atomic access per file as a central system so that printout is not randomly interleaved.

Transparencies

Replication Transparency If the system provides replication for availability or performance reasons, it should not concern the user (as for all transparencies, we include the applications programmer as a user)

Fault Transparency If software or hardware failures occur, these should be hidden from the user.
This can be difficult to provide in a distributed system, since partial failure of the communications subsystem is possible, and this may not be reported. As far as possible, fault transparency will be provided by mechanisms that relate to access transparency. However, when the faults are inherent in the distributed nature of the system, then access transparency may not be maintained. The mechanisms that allow a system to hide faults may result in changes to access mechanisms (e.g. access to reliable objects may be different from access to simple objects). In a software system, especially a networked one, it Is often hard to tell the difference between a failed and a slow running process or processor. This distinction is hidden or made visible here.

Transparencies

Location Transparency The details of the topology of the system should be of no concern to the user. The location of an object in the system may not be visible to the user or programmer. This differs from access transparency in that both the naming and access methods may be the same. Names may give no hint as to location.

Migration Transparency If objects (processes or data) migrate (to provide better performance, or reliability, or to hide differences between hosts), this should be hidden from the user.

Transparencies

Performance Transparency The configuration of the system should not be apparent to the user in terms of performance. This may require complex resource management mechanisms. It may not be possible at all in cases where resources are only accessible via low performance networks.

Scaling Transparency A system should be able to grow without affecting application algorithms. Graceful growth and evolution is an important requirement for most enterprises. A system should also be capable of scaling down to small environements where required, and be space and/or time efficient as required.

Transparencies in a Distributed System

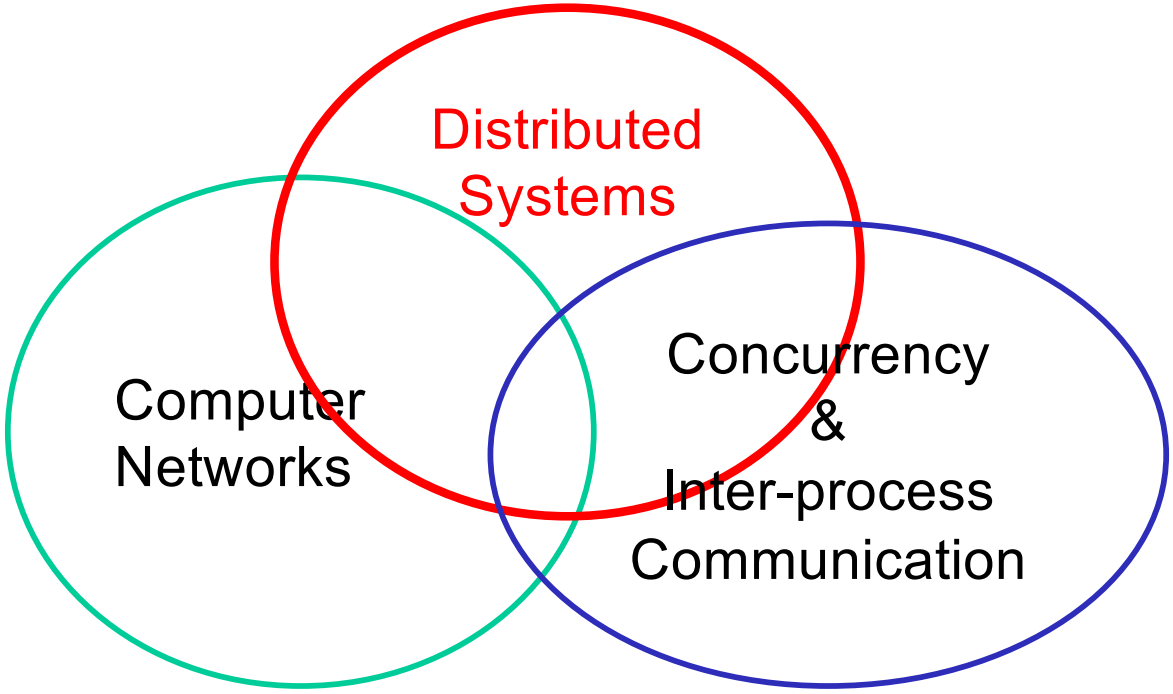
Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Areas

Distributed systems hide network problems …

… but in order to understand and develop them, we need competencies in

- Computer networks and Internet
- Concurrency & Inter-process communication



Two consequences of distribution

- Data travel at (max) the speed of light, but usually the speed is quite slower
- Independent things fail independently

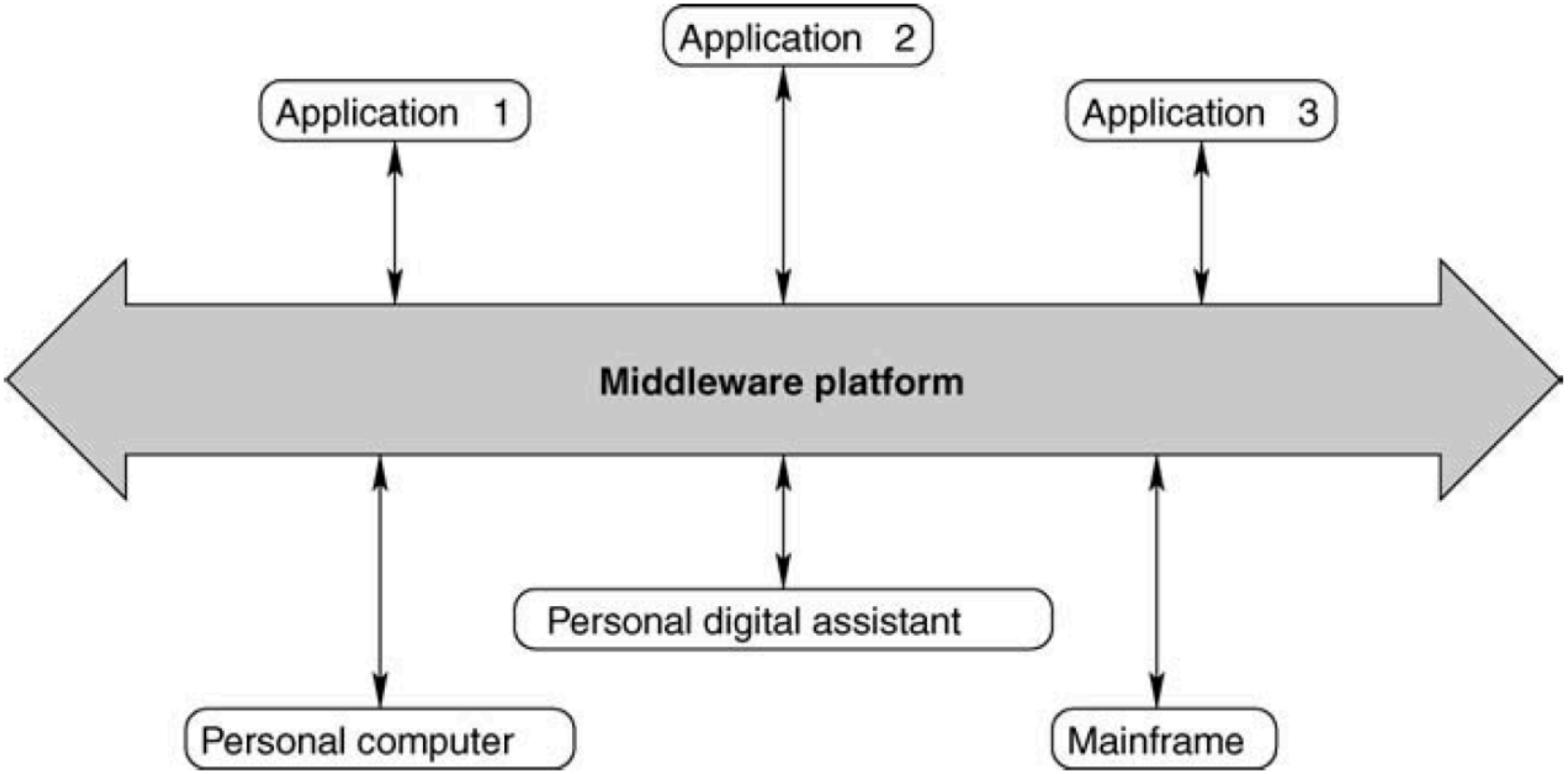
Building distributed systems means dealing with space, time, and having more than one active computing element.

These constraints allow several possible system designs, and after this course you'll have a better idea of how distance, time, and consistency models interact.

Languages for distributed computing

- There are problems in distributed system environments that do not exist in single-machine environments.
- Partial failure, concurrency, and latency are three problems that make distributed computing fundamentally different from local computing

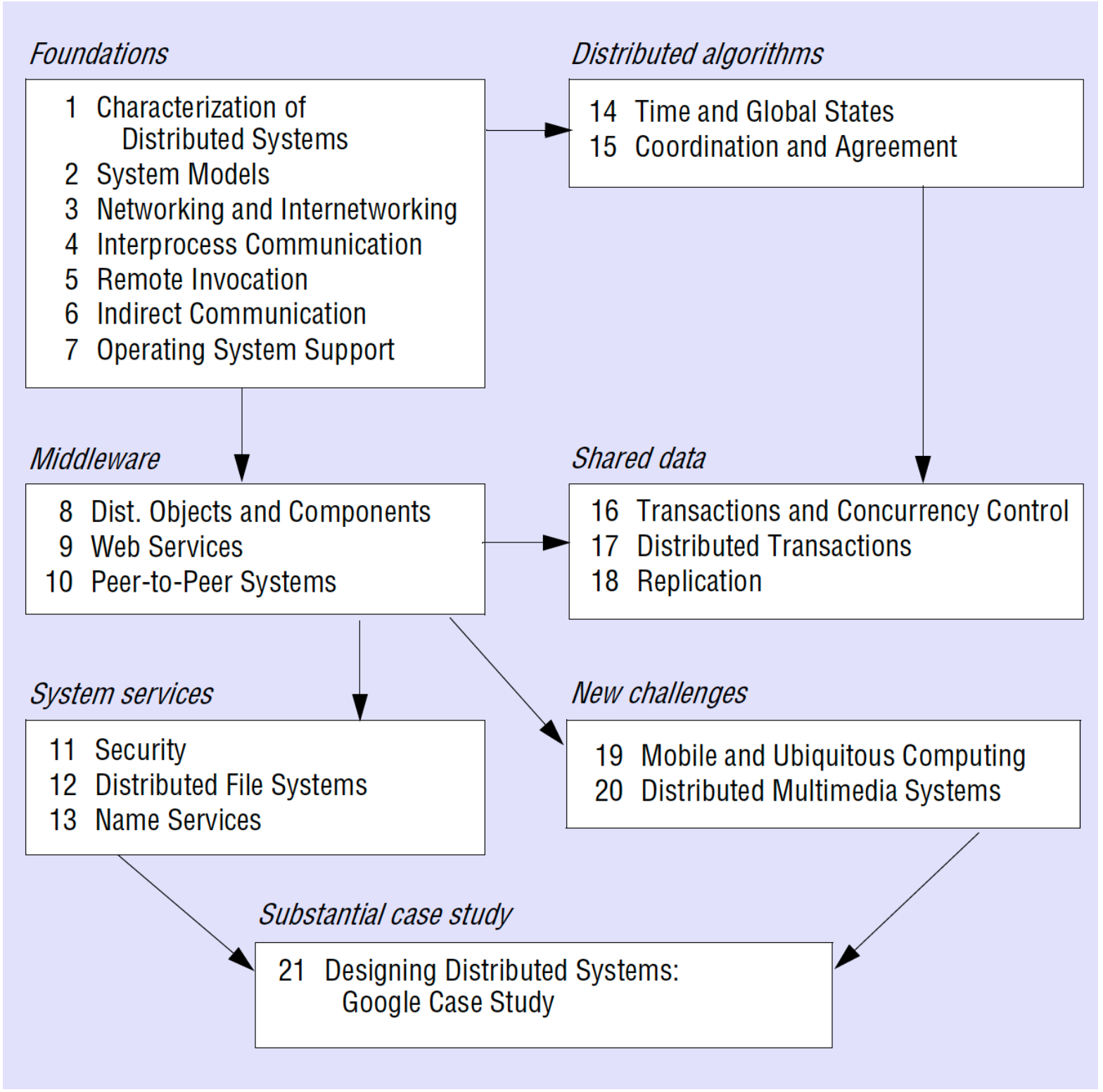
Middleware



Middleware as an infrastructure for distributed systems

Comparing centralized with distributed systems

Criteria	Centralized	Distributed
Economics	low	high
Availability	low	high
Complexity	low	high
Consistency	simple	difficult
Scalability	por	good
Technology	homogeneous	heterogenous
Security	high	low


Foundations
1 Characterization of Distributed Systems
2 System Models
3 Networking and Internetworking
4 Interprocess Communication
5 Remote Invocation
6 Indirect Communication
7 Operating System Support
Distributed algorithms
14 Time and Global States
15 Coordination and Agreement
Middleware
8 Dist. Objects and Components
9 Web Services
10 Peer-to-Peer Systems
Shared data
16 Transactions and Concurrency Control
17 Distributed Transactions
18 Replication
System services
11 Security
12 Distributed File Systems
13 Name Services
New challenges
19 Mobile and Ubiquitous Computing
20 Distributed Multimedia Systems
Substantial case study
21 Designing Distributed Systems: Google Case Study

Clusters of topics

Introduction to distributed systems; RPC Avantages and challenges of distributed systems; “middleware”; transparency goals; client-server systems; failures and retry semantics (all-or-nothing; at-most-once; at-least-once). Remote procedure call (RPC); marshalling; interface definition languages (IDLs); SunRPC; external data representation (XDR).

Network File System and Object-Oriented Middleware Network File System (NFS). Object-oriented middleware (OOM); Corba ORBs, IDL; DCOM.

Practical RPC systems; clocks Remote method invocation (RMI); remote classes vs. serialisable classes; distributed garbage collection; XML-RPC; SOAP and web services; REST. Physical clocks; UTC; computer clocks; clock synchronisation.

Clock synchronisation; logical clocks Clock drift and compensation; Cristian’s Algorithm; Berkeley Algorithm; Network Time Protocol (NTP). Logical time, “happens-before”; Lamport clocks.

Consistent cuts, process groups, and mutual exclusion Consistent global state. Distributed mutual exclusion; central lock servers; token passing; totally ordered multicast.

Elections, consensus, and distributed transactions Leader elections; ring-based algorithm; the Bully algorithm. Consensus. Distributed transactions; Replication and consistency.

Replication in distributed systems, CAP, case studies quorum systems; weak consistency; FIFO consistency; eventual consistency; Amazon’s Dynamo; session guarantees; Consistency, Availability and Partitions (CAP); Google datacentre technologies (MapReduce).

Syllabus (tentative)

Introduction to distributed systems

Principles of programming concurrent distributed systems

Object oriented middleware

Middleware frameworks and technologies

Exam

- Attending class 30%
- Midterm report & presentation: 20%
- Project and final report 50%

Mid-term presentation

Possible topics:

- Middleware for distributed applications
- Important examples of distributed systems
- Fundamental issues in distributed computing

- you have to deliver a small paper (five pages) and a presentation (at least 15 slides, talk 20')

Project

Project Specs

You will implement a prototype of a distributed game with several players (>3).

Main features of the prototype are:

shared state among the players, that depends on the kind of game. As an example it could be the state of a cardboard, the card deck, played cards, … .

Suggested: Java Language and RMI (NOT sockets).

alternative: Python and Celery (or other middleware)

Reliable communications (i.e. no faults; the only kind of fault of the processes is *crash*, otherwise they are fine).

Your system will tolerate at least two processes *crash*; at most all the processes except the winner (the sole survivor).

number of players: larger than 3.

Distributed architecture, the processes are peers; however the registration service could be the a -unique- point of centralization (recommended).

Report on the project

Documentation

A document of at least ten pages; to be sent to me before the exam.

This is its general structure:

- **Abstract** (at most 10 lines): presents the project.
- **Introduction**: sketch of the problem, goals, and state of the art.
- **Main features**: description of: big picture of the project, main problems and their solutions, abstract architecture.
- **Implementation**: diagrams and details on the implementation. Recommended: UML activity diagram about the interactions among the game objects. Also suggested: class, sequence, component, deployment diagrams. Give info on languages/API you used, LOC/#classes you developed, and tests you performed
- **Evaluation**: how did you measure the performances?
- **Conclusions**: concluding remarks and possible improvements.

This course: Scheduling

August 20 16.00-20.50
August 27 16.00-20.50
Sept 3 16.00-20.50
Sept 10 16.00-20.50
Sept 17 16.00-20.50
Sept 23 17:40-20:50
Sept 24 16.00-20.50

Sept 24 midterm presentation (choose topic by sept 14)
Oct 16 (tentative): final exam (demo project, deliver report in advance)

Channels

- **email**: `paolo.ciancarini@unibo.it`
- **Telegram** `@PaLoCaPa (personal)`
- **skype**: `paolociancarini`
- **Linkedin:** search me!
- **Twitter**: `@paolociancarini`

- **Moodle**

Textbooks

vanSteen and Tanenbaum, *Distributed Systems*, 3ed, 2017

Burns, *Designing distributd systems*, O'Reilly, 2018

Etzkorn, *Introduction to middleware*, 2017

Colouris *Distributed systems Concepts and Design*, 5ed 2012

Additional material (slides, papers) distributed during the course

https://www.distributedsystemscourse.com

A repository: http://christophermeiklejohn.com/distributed/systems/2013/07/12/readings-in-distributed-systems.html

A resource on distributed programming

http://dist-prog-book.com/chapter/1/rpc.html

http://dist-prog-book.com/chapter/2/futures.html

http://dist-prog-book.com/chapter/3/message-passing.html

http://dist-prog-book.com/chapter/4/dist-langs.html

http://dist-prog-book.com/chapter/5/langs-extended-for-dist.html

http://dist-prog-book.com/chapter/6/acidic-to-basic-how-the-database-ph-has-changed.html

http://dist-prog-book.com/chapter/7/langs-consistency.html

http://dist-prog-book.com/chapter/8/big-data.html

http://dist-prog-book.com/chapter/9/streaming.html

Ref: https://heather.miller.am/teaching/cs7680/

https://github.com/heathermiller/dist-prog-book

Questions?